UNIVERSIDADE FEDERAL DO PAMPA
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

MATHEUS HENRIQUE KOCH

AVALIANDO OS IMPACTOS DO USO DE VARIANTES PARA OS ALGORITMOS LRU E LFU NO CONTEXTO DE SISTEMAS VIDEO SOB DEMANDA.

Bagé
2022

MATHEUS HENRIQUE KOCH

AVALIANDO OS IMPACTOS DO USO DE VARIANTES PARA OS ALGORITMOS LRU E LFU NO CONTEXTO DE SISTEMAS VIDEO SOB DEMANDA.

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Engenharia de Computação como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Orientadora: Bruno Silveira Neves

Bagé
2022

Ficha catalográfica elaborada automaticamente com os dados fornecidos pelo(a) autor(a) através do Módulo de Biblioteca do Sistema GURI (Gestão Unificada de Recursos Institucionais).

K76 Koch, Matheus Henrique

Avaliando os impactos do uso de variantes para os algoritmos LRU e LFU no contexto de sistemas Video sob demanda. / Matheus Henrique Koch.

- 2022.

80 f.: il.

Orientadora: Bruno Silveira Neves

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Pampa, Campus Bagé, Bacharelado em Engenharia de Computação, 2022.

1. Taxa de acertos. 2. Algoritmos de cacheamento de vídeo. 3. Streaming de video. 4. LRU. 5. LFU. I. Bruno Silveira Neves. II. Título.

SERVIÇO PÚBLICO FEDERAL

MINISTÉRIO DA EDUCAÇÃO

Universidade Federal do Pampa

MATHEUS HENRIQUE KOCH

AVALIANDO OS IMPACTOS DO USO DE VARIANTES PARA OS ALGORITMOS LRU E LFU NO CONTEXTO DE SISTEMAS VIDEO SOB DEMANDA.

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 12 de Agosto de 2022.

Banca examinadora:

_____

Prof. Dr. Bruno Silveira Neves
UNIPAMPA

_____

Prof. Dr. Fábio Luís Livi Ramos
UNIPAMPA

_____

Prof. Dr. Carlos Michel Betemps
UNIPAMPA

Assinado eletronicamente por **FABIO LUIS LIVI RAMOS**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 19/08/2022, às 16:36, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **BRUNO SILVEIRA NEVES**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 19/08/2022, às 16:37, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **CARLOS MICHEL BETEMPS**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 19/08/2022, às 16:55, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0902128** e o código CRC **6F0D0374**.

Referência: Processo nº 23100.017423/2022-50 SEI nº 0902128

RESUMO

O relatório da Cisco (2020) apresenta uma previsão sobre o desenvolvimento da Internet em geral, tendo em vista que será necessária uma evolução em todos os níveis de *Internet Service Providers* (ISPs) a fim de conseguir suprir a demanda total de banda. Já o relatório de Cullen e Cantor (2020) mostra que o tráfego de vídeo é responsável por mais de 50% do tráfego da rede, e com essa informação, é possível entender que o crescimento e a otimização das Redes de Distribuição de Conteúdo, incluindo um dos seus elementos mais importantes que são os servidores de vídeo do tipo *Proxy*, pode tornar-se crucial para dar suporte a esse crescimento. Os algoritmos usados dentro dos servidores *proxy* de vídeo representam uma importante parte do desempenho desse servidor, pois impactam no aumento da taxa de acertos desses algoritmos, aumento este que pode representar uma diminuição do tráfego em partes centrais da rede, como seu *backbone*. Neste contexto, o objetivo deste trabalho consiste em avaliar os benefícios produzidos pela modificação dos algoritmos *Least Recently Used* (LRU) e *Least Frequently Used* (LFU) buscando um aumento da taxa de acertos através da implementação de uma janela temporal fixa de análise. Conforme resultados deste trabalho, a implementação desta janela produziu uma melhora de mais ou menos 20% na taxa de acertos do LFU para cenários com número Zipf alto. Essa taxa pode variar dependendo do cenário e do tamanho da janela que forem usados. Para simulação destes algoritmos para foi desenvolvido um simulador especificamente para o presente trabalho, o qual consegue simular um ambiente a fim de permitir a análise da taxa de acertos para diferentes cenários de carga, modelados através do uso da Distribuição de Poisson para geração do intervalo entre entradas dos clientes e da Lei de Zipf para descrever a frequência ou popularidade dos vídeos do acervo. Inclusive foram realizadas simulações para os algoritmos Randon, LRU, LFU, *First In First Out* (FIFO), CC e CARTE com objetivo de contrapor os resultados encontrados.

Palavras-chave: Taxa de acertos; Algoritmos de cacheamento de vídeo; Streaming de video; LRU. LFU.

ABSTRACT

The CISCO report (2020) presents a forecast about the development of the Internet in general, considering that it will be necessary an evolution at all levels of Internet Service Providers networks (ISPs) in order to be able to supply the total demand. The report by Cullen Cantor (2020) shows that the video traffic is responsible for more than 50% of network traffic and with this information it is possible to understand that the growth and optimization of the video-on-demand (VoD) proxy server can become crucial. The algorithms used within the VoD proxy servers represent an important part of the performance of this server, as they generate an increase in the success rate of these algorithms, an increase that may represent a decrease of traffic in parts of the network. Performing a simple modification in the Least Recently Used (LRU) and Least Frequently Used (LFU) algorithms can bring an increase in the success rate, such as the implementation of a fixed analysis time window, which brought an improvement around 20% in the LFU hit rate for scenarios with high Zipf number. This rate may vary depending on the scenario and the size of the window being used. The scenario used in this work was generated using the Poisson Distribution for the customer input interval and the Zipf Law that describes the frequency of the videos. A simulator developed specifically for the present work was used, which manages to simulate an environment in order to achieve a correct rate of scenarios without having all the memory and bandwidth of the simulation. Simulations were even performed for the Randon, LRU, LFU, First In First Out (FIFO), CC and CARTE algorithms in order to compare the results found.

Keywords: Hit rate, Video caching algorithms, Video streaming, LRU, LFU .

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
ACM	Association for Computing Machinery
IEEE	Institute of Electrical and Electronics Engineers
UNIPAMPA	Universidade Federal do Pampa
UFRGS	Universidade Federal do Rio Grande do Sul
Proxy	Servidores proxy
Proxy VoD	Servidores proxy video sob demanda
LFU	Least Frequently Used
LRU	Least Recently Used
VoD	Video on Demand
ZC	Coeficiente Zipf
DV	Duração dos Vídeos
DS	Duração das Simulações
LB	Largura de Banda Máxima para o canal servidor-proxy
NC	Número de Clientes
NV	Número de Vídeos
RAM	Random Access Memory
SIMPRO	Simulador de proxy
TM	Tamanho da Memória
QoE	Quality of experience
ISPs	Internet Service Providers' networks
CDNs	Content Delivery Networks
BLOCO	um segundo de Video
DDos	Distributed Denial of Service

SUMÁRIO

1 INTRODUÇÃO

Com o passar dos anos, podemos notar a evolução tecnológica que vem trazendo melhorias na capacidade de armazenamento, de processamento, de banda larga e de memória. Assim como o surgimento de novos métodos de codificação e decodificação de áudio e vídeo, assegurando uma alta qualidade nos fluxos multimídia distribuídos. Com esses avanços, as aplicações de Vídeo sob Demanda (*Vídeo on Demand* - VoD) têm ganhado cada vez mais espaço na Internet.

A principal função do Vídeo sob Demanda é a distribuição de vídeos para diversas aplicações, entre elas aquelas relativas ao entretenimento, educação e informação. Tais aplicações podem ser executadas através de diferentes dispositivos como TVs, telefones móveis e computadores, e também possibilitam o acesso às operações de pausa, troca de idioma, avançar e a versatilidade de assistir o vídeo no momento que desejar sem a necessidade de esperar baixar todo o vídeo para iniciar sua exibição.

As informações contidas no relatório Cisco (2020), o qual traz uma previsão sobre o crescimento do uso de Internet no período de 2020 a 2023, nos dão a noção do quanto de esforços em termos de desenvolvimento estrutural e tecnológico é necessário para suprir a futura demanda por *streaming* de vídeo. Para gerar as previsões do relatório Cisco (2020), primeiro foi obtido a taxa de crescimento real dos anos de 2018 e 2019 para cada aspecto. Em seguida são pegos os dados de 2019 e multiplicados pela taxa de crescimento obtida para cada aspecto, assim teremos a previsão dos dados de 2020. Essa procedimento foi realizado até o ano de 2023, sempre utilizando os dados do ano anterior e multiplicando pela respectiva taxa de crescimento, é uma previsão de curto período que tem uma margem de erro baixa, comparada a previsões de maior tempo.

O principal dado exposto pelo relatório da Cisco (2020) é a previsão do crescimento de usuários da Internet de 51% da população global em 2018, que corresponde em torno de 3,9 bilhões de usuários, para 66% da população de 2023, que corresponderia aproximadamente a 5,3 bilhões de usuários. Estima ainda que a velocidade de banda larga fixa mais do que dobrará até 2023, em comparação a 2018, e assim como o número de aparelhos de TV de tela plana conectados em UHD ou 4K terá um acréscimo de 66%. Além disso, expõe sobre o aumento global do número de dispositivos e conexões per capita.

No relatório de Cullen e Cantor (2020), há informações específicas sobre as principais categorias e aplicações que consomem o tráfego global de Internet. Uma das

primeiras constatações observadas é que mais de 70% do volume de tráfego global é composto por vídeos, jogos e redes sociais. Dentro dessa porcentagem, o *streaming* de vídeo soma 57%, sendo que esse valor é composto por diferentes aplicações que usam vídeos, sendo as principais entre elas o YouTube com 15% e a Netflix com 11%.

O relatório de Cullen e Cantor (2020) traz a confirmação que as aplicações de Vídeo sob Demanda tiveram um crescimento do ano 2019 para 2020. Reunindo as informações contidas nos relatórios Cisco (2020) e Cullen e Cantor (2020), temos evidências para acreditar que as aplicações de Vídeo sob Demanda (VoD) terão cada vez mais espaço na Internet.

A pandemia mundial de 2020 influenciou nos resultados dos dois relatórios, pelos motivos da redução da mobilidade urbana, do aumento do trabalho remoto e do isolamento social, causando o crescimento do uso da Internet para diversos fins. Os autores Cullen e Cantor (2020) trazem alguns dados sobre o uso intensivo da internet nos primeiros meses da pandemia, do dia 1 de fevereiro até 19 de abril de 2020 houve um aumento de 40% no tráfego de internet. Esse aumento não ocorreu só em alguns horários mas constantemente em todo dia e ele foi absorvido sem grandes impactos no desempenho da rede e Qualidade do Serviço (*Quality of Service* - QoS).

1.1 Estrutura de Redes e Problema do tráfego de vídeo

Com as informações fornecidas pelo relatório Cisco (2020), entende-se que será necessário um aprimoramento das Redes de Distribuição de Conteúdo (*Content Delivery Networks* - CDNs). As CDNs são um conjunto de servidores de amplas variedades e são compostos por milhares de servidores, distribuídos geograficamente pelo mundo, que são capazes de sustentar grandes cargas de trabalho e pontos críticos de tráfego. (BUYYA; PATHAN; VAKALI, 2008)

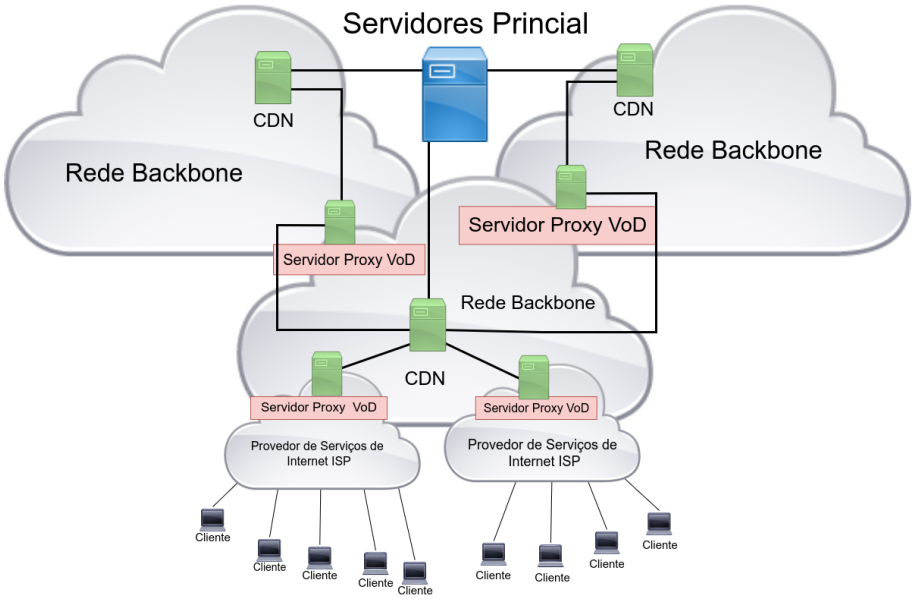
Entre os tipos de servidores encontrados em cada CDN, observa-se o tipo *Proxy* (também chamados de Servidores Substitutos) e os Data Centers, que armazenavam conteúdos estáticos como códigos *Hypertext Markup Language* (HTML) e imagens. Atualmente, as CDN também podem armazenar e distribuir conteúdos dinâmicos, como videoconferências e live *streaming*, além de promover proteção contra ataques de *hackers*, *spammers* e ataques de *Distributed Denial of Service* (DDoS). (BUYYA; PATHAN; VAKALI, 2008)

As CDNs são interessantes para os provedores de conteúdo porque a

responsabilidade pela hospedagem de conteúdo é colocada para a infraestrutura CDN que replica conteúdo entre os núcleos de servidores espalhados pelo mundo e isso faz com que o conteúdo publicado esteja altamente disponível. As empresas como Cloudflare, Akamai, Encapsula, MaxCDN e Google Cloud Platform são grandes empresas que atuam em diferentes níveis de fornecimento de serviços pela Internet e elas fornecem suporte em nível de CDNs em produtos oferecidos. (BUYYA; PATHAN; VAKALI, 2008)

Os servidores de *proxy* são servidores intermediários que fazem a conexão dos clientes com o servidor principal, e podem ter outras funções como servir para monitoramento, filtragem, diminuição de tempo de resposta, segurança, etc. É possível encontrar servidores *proxy* distribuídos em diferentes pontos do backbone, assim como nos Provedores de Serviços de Internet (*Internet Service Provider*-ISP) e nas CDNs. O aumento do número de servidores *proxy* nos *backbones* podem criar um melhor suporte para o desenvolvimento da internet previsto no relatório do CISCO. Esse servidor *proxy* pode funcionar como cache de informações e dados, reduzindo a quantidade de tráfego nos diferentes níveis da rede e permitindo a resposta aos clientes com menor latência. Essa diminuição do tempo de resposta para ocorre devido à distância entre cliente e a fonte de dados se tornar menor. Esses *proxies* são localizados nas periferias dos *backbones* onde estão as ISP, como pode ser visto na Figura 1

Figura 1 – Servidor Principal conectado com CDN e Servidor *proxy* nas margens dos *backbones*.



Fonte: Autoria própria, mas ícones usados são fornecidos pela (CISCO, 2016)

As CDNs e servidores de *proxy* se tornam fundamentais para a QoS disponibilizada aos usuários, pois o uso de CDN com servidores *proxy* ajuda principalmente a minimizar os atrasos no carregamento de conteúdo, reduzindo a distância física entre o servidor e o usuário. Além disso, o conteúdo replicado integralmente ou parcialmente possibilita o aumento da disponibilidade frente a um número crescente de clientes, preservando assim a sua QoS.

As melhorias proporcionadas pelo uso de uma CDN podem ser entendidas ao analisar uma rede sem CDN e com CDN. Em uma rede sem CDN pode ser imaginada na qual os clientes conectam-se diretamente no servidor principal, isso implica que os clientes sempre percorram uma longa distância para conseguir as informações desejadas e também será responsável por atender todas as requisições dos clientes. Já uma rede com CDN, como na Figura 1, encontram-se os servidores *proxy* e CDN no meio do caminho entre os clientes e o servidor principal. Com isso, além da distância entre o cliente e o servidor video ser menor, a carga da rede e a carga de cliente será menor sobre o servidor principal, pelo fato de que as requisições dos clientes não necessitam chegar até o servidor principal(o servidor principal em cada solicitação), pois suas demandas podem ser supridas em parte ou totalmente pelos *proxies* e CDN.

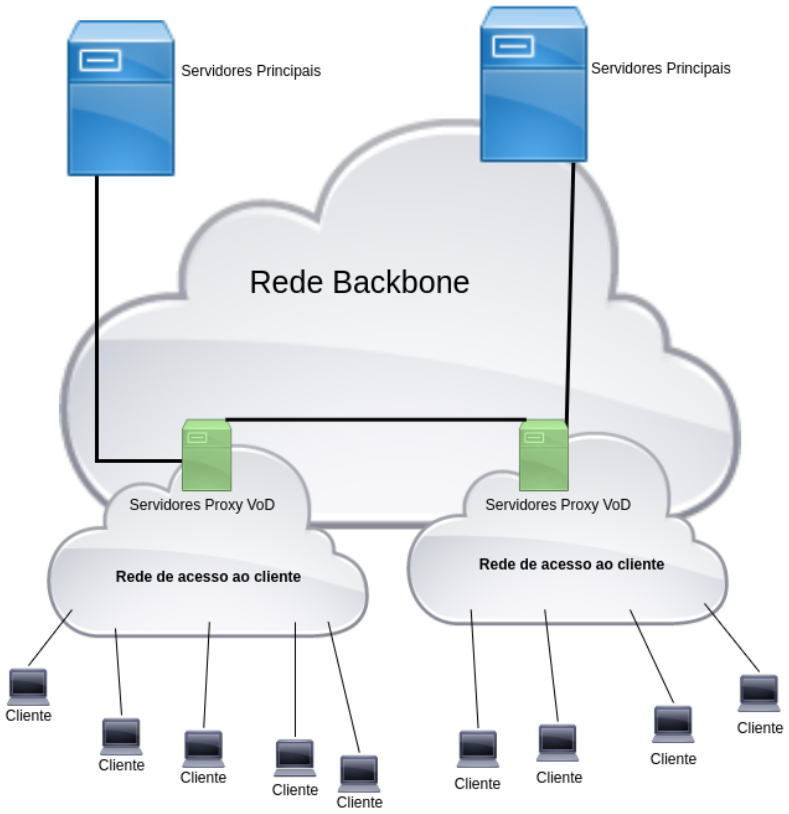
1.2 Escopo da proposta de trabalho

Conforme White et al. (2008), os servidores do tipo *proxy* para Vídeo sob Demanda implementam especificamente a função de cacheamento de vídeos, assim armazenando vídeos completos ou partes de vídeos. Para realizar essa função principal, outras funções secundárias são executadas pelo *proxy* VoD, como liberar memória automaticamente, para armazenar as novas partes de vídeo provenientes do servidor principal, a partir das solicitações de vídeos realizadas pelos clientes. Para atender essa função, os *proxies* VoD normalmente têm memória de armazenamento maior que os demais servidores *proxy*, isto porque os vídeos frequentemente têm tamanhos maiores do que o de outros tipos de dados, como uma página web ou uma foto.

Na Figura 2, de maneira muito simplificada, é representado o funcionamento da rede com os servidores *proxy* VoD. Eles estão localizados no meio do caminho, de forma que cada requisição de cliente, assim como o correspondente fluxo de vídeo de retorno, não precisa percorrer todo o caminho entre o cliente e o servidor principal, pois o servidor *proxy* VoD é quem precisará se comunicar com o servidor principal sempre que o *proxy*

não dispuser de recursos em sua cache local para suprir a demanda dos seus clientes. Assim, caso um ou mais clientes do servidor estiverem assistindo a um mesmo vídeo, a carga para o servidor principal tenderá a ser menor em razão do compartilhamento de conteúdo ocorrido no âmbito do *proxy* VoD, ou seja, os clientes serão abastecidos com dados de vídeos trazidos para a memória do *proxy* em um momento anterior para abastecer as primeiras requisições realizadas para estes mesmos vídeos.

Figura 2 – Servidor *proxy* VoD nas margens da redes, para conectar o servidor principal de vídeo com os Clientes.



Fonte: Autoria própria, mas ícones usados são fornecidos pela (CISCO, 2016)

Desta forma, os *proxies* VoD se tornaram importantes nas CDNs, pois possibilitaram o melhor funcionamento do cacheamento. Antes, os algoritmos de cacheamento de uso geral tinham de escolher se ocupariam os recursos disponíveis de memória para armazenamento de uma parte de vídeo ou se usariam a mesma quantidade de recursos para armazenar vários outros tipos de arquivos. Essa diferença entre tamanhos de arquivos de vídeo e demais arquivos pode ser observada no consumo de banda apresentado no relatório de Cullen e Cantor (2020), onde fica evidente que as empresas que oferecem serviços de distribuição de vídeo são as maiores responsáveis pelo tráfego da rede. Assim, a disponibilidade do *proxy* VoD entre o cliente e a CND e também dentro

da CND contribui para a redução do número de clientes que o servidor principal precisa abastecer diretamente.

É possível observar diferentes configurações de hardware nas implementações dos *proxys* VoD em uso, contudo algumas características do hardware costumam ser comuns nestas implementações como, por exemplo, possuir grande capacidade de armazenamento. Essas distinções do hardware são causadas pela responsabilidade de cacheamento do tipo de arquivo, já que arquivos de vídeo têm tamanhos maiores e, muitas vezes, dependentes da qualidade de vídeo que se pretende entregar aos clientes. Para que os recursos de hardware sejam bem usados, é necessário configurar uma política de substituição adequada para cada *proxy* e, na literatura, podemos encontrar diversos tipos de algoritmos de substituição que são usados em diferentes locais. Segundo Gagne (2015), pode-se encontrar diferentes algoritmos de substituição em diferentes contextos de uso, entre eles o *Least Frequently Used* (LFU) e o *Least Recently Used* (LRU).

O LRU e LFU são algoritmos que podem ser encontrados em grande parte dos microprocessadores de propósito geral que utilizam esta abordagem como, por exemplo, os processadores AMD Opteron (PATTERSON; HENNESSY; ASHENDEN, 2007) e Fujitsu Sparc64 V (IKEDA et al., 2004). Também pode ser encontrada na hierarquia de memória (PATTERSON; HENNESSY; ASHENDEN, 2007), (GAGNE, 2015). Tanto o LRU quanto o LFU são algoritmos de substituição amplamente conhecidos e permitem ser empregados em diferentes problemas, no entanto, nem sempre seus resultados serão melhores para todos cenários. A busca por resultados melhores tem influência no desenvolvimento de novos algoritmos de substituição, no qual podemos usar algoritmos conhecidos como base do desenvolvimento dos novos. Na literatura, é possível encontrar várias derivações dos algoritmos LRU e LFU como o Segmented LRU (SLRU) de Karedla, Love e Wherry (1994), *Least frequent recently used* (LFRU) de Bhat et al. (2018) e LFU *with dynamic aging* (LFUDA) do Jayarekha e Nair (2010).

Neste trabalho, pretende-se avaliar os benefícios e custos inerentes ao uso de versões modificadas das políticas de substituição de cache LFU e LRU para uso em *proxies* VoD, incluindo uma janela de tempo para limitar o espaço de análise dos algoritmos, buscando identificar se um olhar de mais curto prazo tende a contribuir para o aumento da eficiência dos mesmos. Esses algoritmos serão usados em um simulados de servidor *Proxy* VoD, onde ocorrerá a avaliação de seu rendimento com essas modificações. No simulador construído, são utilizadas as distribuições de Poisson, para definir o intervalo de chegada no cliente, e a distribuição Zipf, para atribuir a

popularidade para cada vídeo do acervo, com o intuito de se gerar as cargas de testes para simular vários cenários de comportamento dos usuários do sistema VoD. Um fator importante para a análise do desempenho dos algoritmos de substituição é a taxa de acertos, haja vista que essa taxa causa impacto importante na redução do tráfego de rede e uma melhor QoS para os clientes. Para o teste e validação, propomos criar diferentes cenários e diferentes configurações *proxys* para podermos encontrar uma média da taxa de acertos para cada algoritmo proposto e depois comparar a média da taxa de acertos com algoritmos presentes na literatura como LFU e LRU.

O restante deste texto está organizado em 5 capítulo. No primeiro capítulo, são apresentados os algoritmos de armazenamento temporário, trazendo alguns trabalhos que contemplam os algoritmos LRU e LFU e depois suas definições e funcionamentos e também representação do comportamento de rede. No segundo capítulo, são apresentadas as variantes para LRU e LFU, desenvolvidas para tentar prover aumento da taxa de acertos em cenários onde os algoritmos clássicos LFU e LRU vêm sendo aplicados. No Terceiro capítulo, ambiente experimental, mostraremos a motivação da construção, o funcionamento, os parâmetros padrões e a estratégia adotada para validação do simulador desenvolvido para avaliar as variantes propostas neste estudo. No quarto capítulo, são apresentados o ambiente experimental e os resultados dos experimentos. No quinto capítulo, são apresentadas as considerações finais deste trabalho.

2 ALGORITMOS DE ARMAZENAMENTO TEMPORÁRIO

Neste capítulo, é feita uma revisão sistemática da literatura. Na seção 2.1, pretende-se fazer uma busca e filtrar por trabalho que esteja relacionado ao assunto que será debatido neste trabalho. Depois disso, na seção 2.2, abordaremos os algoritmos usados como alvo para os aprimoramentos propostos neste trabalho, LRU e LFU, assim como outros algoritmos clássicos e também mais avançados (como o CC) que serão utilizados em uma análise comparativa com as alternativas propostas, tais como o Random e o FIFO. No fim deste capítulo, falaremos sobre as duas distribuições usadas para gerar de carga.

2.1 Revisão sistemática da literatura

2.1.1 Método de levantamento bibliográfico

O trabalho de Brizola e Fantin (2016) traz algumas justificativas de como usar um método de revisão sistemática da literatura. E entre alguns desses pontos, está a orientação para execução de uma busca organizada que pode ser usada em diferentes temas, como encontrar os vieses do tema que não foram abordados, além disso, ajuda na redução e organização das informações buscadas e consequentemente terá um refinamento do resultado da sua pesquisa.

Para iniciar as buscas por bibliografias para o trabalho, seguimos o guia de revisão sistemática da literatura apresentada por Neiva (2016). O referido guia traz traz 10 passos para fazer essa revisão, sendo que alguns destes passos foram adaptados. Neste guia, a autora de maneira simples e didática mostra os passos para fazer uma boa revisão sistemática para área da computação.

O trabalho de Neiva (2016) descreve 10 etapas, sendo que cada etapa gera um resultado que é usado para a etapa subsequente. Abaixo, estão os títulos de cada etapa e os resultados obtidos.

1. **Definir as questões de pesquisa principais para a revisão:**
   Aplicações de algoritmos LRU, LFU e derivados em servidores *proxy* VoD Demanda? Como são usados?
2. **Definir as palavras-chave:**

- *video caching management policy*
- *video caching replacement algorithm LRU LFU windowing window*
- *Surrogate video server LRU LFU windowing window*
- *proxy VoD LRU LFU windowing*
- *video-on-demand proxy server*
- *multimedia streaming caching*
- *LRU proxy VoD*
- *LFU proxy VoD*
- *windowing LFU LRU*
- *window LFU LRU*

3. **Definir a string de busca:**
   (streaming OR multimedia OR video) AND proxy AND (VoD OR caching OR video-on-demand) AND ((management OR replacement OR surrogate) AND policy) AND ((LRU OR LFU) AND proxy VoD) AND (windowing OR window)
4. **Definir as bases de busca:**
   - Google Acadêmico (VERSTAK, 2004) ;
   - IEEE Xplore (ELETRICISTAS, 1963);
   - ACM (HAMMING, 1947);
   - *Springer Link* (SPRINGER, 1842);;
5. **Refinamento da string:**
   streaming AND proxy AND (VoD OR caching OR video-on-demand) AND ((management OR replacement OR surrogate) AND policy) AND ((LRU OR LFU) AND proxy VoD)
6. **Execução da string de busca nas bases:**

Tabela 1 – Resultado da busca em sites de trabalhos acadêmicos:

Base de busca	*Número de trabalhos*
Google Acadêmico	199
IEEE Xplore	12
ACM	77
Springer Link	6

Fonte: Autoria própria

7. **Baixar e armazenar o resultado das buscas:**

Foi utilizada a ferramenta Mendeley de Foeckler Victor Henning (2008) para gerenciar as referências, de forma que os resultados sejam centralizados, filtrados e armazenados. Armazenamos todos os resultados da etapa 6 na ferramenta Mendeley e, ao usar a extensão Mendeley no *Chrome*, isso causou a entrada de alguns artigos duplicados no resultado, influenciando o aumento de arquivos no gerenciador para 306. Essa falha tende a não impactar no resultado final do processo de seleção de itens bibliográficos, apenas na quantidade de trabalho necessário para se retirar os arquivos duplicados.

8. **Definir os critérios de inclusão e exclusão:**
   Como critério de seleção, buscou-se identificar artigos recentes que fizessem uso dos dois algoritmos explorados neste trabalho. Desta forma, foi possível constatar que os algoritmos LFU e LRU são usados em artigos recentes nas mais diversas aplicações. A exemplo disso, alguns artigos usam estes dois algoritmos como base comparativa para avaliação de novas soluções de cacheamento de vídeo. Por outro lado, artigos recentes também demonstram novas propostas de algoritmos de cacheamento que se baseiam nos LRU e LFU clássicos.
9. **Seleção de artigos:**
   Esta etapa se subdividiu em três partes: na primeira delas, os artigos foram selecionados com base em uma análise de título e resumo apenas; na segunda parte, os artigos foram selecionados com base em uma análise de Introdução e Conclusão; na terceira etapa, os artigos foram selecionados com base em uma análise da leitura completa e verificação de qualidade.
   Ao analisar referências no Mendeley (2007), percebeu-se que alguns trabalhos não tinham correspondentes arquivos em pdf, sendo que foi feita uma pesquisa para tentar encontrar e baixar o arquivo texto dos trabalhos, e, mesmo feita esta busca, observou-se que alguns trabalhos não estão disponíveis abertamente, isto é, com acesso público. Sendo assim, foram removidos os trabalhos para os quais não se obteve completo acesso ao texto.
10. **Extração das respostas relacionadas às questões de pesquisa:**
    A Tabela 2 listas os trabalhos da base de publicações acadêmicas resultado da revisão sistemática da literatura realizada. Em algumas das produções acadêmicas abrangem desde a arquitetura de rede *proxy*s até as configurações de cada *proxy* e outras trazem cálculos de complexidade de diferentes algoritmos. São uma

variedade de publicações que conseguem responder totalmente ou parcialmente as questões de pesquisa. No tópico a seguir “Trabalhos relacionados”, traremos mais uma discussão sobre esses trabalhos selecionados.

Tabela 2 – Trabalhos selecionados da revisão sistemática da literatura

Autor(es)	*Titulo do trabalhos*
Bhat et al. (2018)	*SABR: Network-Assisted Content Distribution for QoE-Driven ABR Video Streaming*
Desmouceaux et al. (2019)	*A Content-aware Data-plane for Efficientand Scalable Video Delivery.*
Ghoreishi (2017)	*Bring the Content Closer to the End User: In-NetworkAdaptation and Caching of Mobile Video*
Koch et al. (2018)	*Category-aware Hierarchical Caching for Video-on-Demand Content onYouTube*
Pal e Kant (2017)	*Neighborhood Aware Caching andInterest Dissemination in Content Centric Networks*
Taher et al. (2018)	*A Review on Cache Replacement Strategies in Named Data Network*
Wang e Chen (2018)	*Cascade mapping: Optimizing memory efficiency forflash-based key-value caching*

Fonte: Autoria própria

2.1.2 Trabalhos relacionados

Essa sessão não fica restrita somente aos trabalhos encontrados através do “Método de levantamento bibliográfico”, em virtude de alguns fatores como período da busca e inexatidão de algumas palavras-chave que causou a não aparição de alguns trabalhos relevantes. Assim, ao longo do desenvolvimento da pesquisa foram encontrados mais trabalhos, os quais complementam a nossa de lista de bibliografia e gerando a tabela 3.

Ao analisar diversos trabalhos, foi possível ver que os algoritmos dos servidores de *proxy* compõem uma pequena parte de uma arquitetura e organização da rede. Podem influenciar muito no desempenho do sistema de distribuição de vídeo, pois os algoritmos de substituição são responsáveis em classificar o conteúdo do servidor *proxy* que será removido da memória com passar do tempo. Um ponto importante para eficiência dos algoritmos de substituição é a taxa de acerto, quanto maior a taxa mais eficiente o algoritmo.

Tabela 3 – Todos os trabalhos selecionados fim da pesquisa

Autor(es)	*Título do trabalhos*
Bhat et al. (2018)	*SABR: Network-Assisted Content Distribution for QoE-Driven ABR Video Streaming*
Desmouceaux et al. (2019)	*A Content-aware Data-plane for Efficientand Scalable Video Delivery.*
Ghoreishi (2017)	*Bring the Content Closer to the End User: In-NetworkAdaptation and Caching of Mobile Video*
Koch et al. (2018)	*Category-aware Hierarchical Caching for Video-on-Demand Content onYouTube*
Pal e Kant (2017)	*Neighborhood Aware Caching andInterest Dissemination in Content Centric Networks*
Taher et al. (2018)	*A Review on Cache Replacement Strategies in Named Data Network*
Wang e Chen (2018)	*Cascade mapping: Optimizing memory efficiency forflash-based key-value caching*
Chen et al. (2019)	*Hit ratio driven mobile edge caching scheme for video on demand services*
Hao et al. (2018)	*Knowledge-centric Proactive Edge Caching Over Mobile Content Distribution Network*
Osman e Osman (2018)	*A comparison of cache replacement algorithms for video services*
Neves (2015)	Proposta de algoritmo de cacheamento para *proxies* VoD e sua avaliação usando um novo conjunto de métricas

Fonte: Autoria própria

Para conseguir fazer as comparações mais justas, deve-se colocar os algoritmos sob a mesma carga de trabalho. Os autores como Bhat et al. (2018), Osman e Osman (2018), Pal e Kant (2017), Taher et al. (2018) e outros, fazem análises através de simulações usando diferentes estratégias de substituição na cache.

Nos trabalhos relacionados, encontra-se o trabalho do Koch et al. (2018), os quais propõem uma estratégia de cacheamento denominada *Adaptive Category-aware Designed Caching (ACDC)*. O diferencial desta estratégia é que ela divide o espaço de armazenamento em categorias de conteúdo para os vídeos, alocando um espaçamento específico para cada tipo de conteúdo. O ambiente de simulação possui uma estrutura com hierarquias de cache e cada nível de cache é dividida nas seguintes categorias: Música, Entretenimento Pessoas, Comédia e Diversos. Nos resultados finais descritos pelo autor, houve um aumento de 18.39% na taxa de acerto em relação aos demais algoritmos analisados no trabalho. Esse resultado demonstra que a variação do conteúdo por categorização influencia a cache, impedindo que a cache aloque espaço exagerado

para um determinado comportamento temporário dos clientes.

Já no trabalho de Bhat et al. (2018), os autores projetam, implementam e avaliam uma nova arquitetura denotada como *SDN-Assisted ABR Streaming* (SABR). Nesta arquitetura, foi adotada a política de substituição LRU para implementação da cache de vídeo no início do projeto, mas para melhorar os resultados eles testaram outro algoritmo de cacheamento como o *Time-to-Live* (TTL). O TTL consiste em colocar tempo de vida máximo para cada bloco ficar na cache e ao comparar os resultados do LRU e TTL, o TTL trouxe um melhor desempenho e propõem usar um TTL adaptativo. O TTL adaptativo consiste em conseguir variar o tempo de vida de cada bloco de acordo com a frequência de acesso ao bloco.

Na obra de Desmouceaux et al. (2019), utilizou-se um algoritmo LRU em dois momentos: antes de salvar algo na cache, é feito uma verificação da popularidade do bloco do vídeo, analisando se o identificador do bloco existe em uma lista limitada compartilhada por conjunto de *proxy*, a fim de verificar se é necessário ou não salvar a cache; e depois de salvo usam na política de substituição LRU. Essa lista de popularidade de blocos usa a política de substituição de LRU para remover o identificador do bloco que não foi usado recentemente, pois a lista é limitada, e toda vez que é feita uma solicitação de um bloco que não foi encontrado na cache é colocado o identificador do bloco na lista popularidade.

O trabalho de Osman e Osman (2018) faz a comparação da taxa de acertos com os seguintes algoritmos: OPT (*Optimal Algorithm*), CC (*chumk-based caching*), QC (*Quality-based*), LRU-2 (LRU-K), LRU (*Least Recently Used*), LFU (*Least Frequently Used*) e FIFO (*First in First out*). Esse do Osman trabalho apresenta aspectos muito parecidos com nosso trabalho, entretanto seus resultados podem apresentar uma distorção em relação aos nossos resultados finais, devido a geração de carga do simulador. O autor Osman desse trabalho opta por usar a distribuição de Zipf para estimar a popularidade de cada requisição de vídeos, e para frequência não é usado o Poisson. Os resultados finais desse trabalho confirmaram os resultados que os algoritmos CC e OPT ficaram na frente na taxa de acertos.

No trabalho do Chen et al. (2019), os autores apresentam uma nova proposta de algoritmo, no qual seu algoritmo considera a otimização conjunta de cache de vídeo e a transcodificação em tempo real. Para contrapor seu algoritmo, os autores comparam com outros algoritmos de cacheamento que são LRU, LFU e WGDSF. Eles utilizaram a plataforma do Matlab para fazer as simulações e para geração dos arquivos de carga,

utilizaram a Zipf 0.6 para popularidade dos vídeos e Poisson 0.9 para intervalo de chegada das requisições. O resultado final mostra que o algoritmo proposto pelo autor apresenta melhores resultados que os outros algoritmos comparados no trabalho.

2.1.3 Simuladores

Dos trabalhos listados na Tabela 3, as informações relevantes sobre as simulações podem ser extraídas e importadas para a Tabela 4. No entanto, algumas informações não puderam ser obtidas porque não estavam claramente especificadas no texto, nesse caso, optou-se em deixar os quadros em branco com um traço dentro.

Era razoável encontrar trabalhos usando simuladores interessantes/importantes como NS-3, SD-WANs, VPP e CCN *routers*, porém são muito mais complexos para trabalhar. Assim, também foi possível ver que alguns trabalhos da Tabela 4 desenvolveram seus próprios simuladores, mas alguns optaram em não disponibilizar o código fonte.

Tabela 4 – Todos os trabalhos selecionados fim da pesquisa

Autor(es)	*Distribuições (popularidade e frequência)*	*Algortimos comparados*	*Simulador*
Chen et al. (2019)	Zipf (0.6) e Poisson (0.9)	PROPOSED, LRU, LFU e WGDSF.	MATLAB
Hao et al. (2018)	-	KCOC e PSA	NS-3
Osman e Osman (2018)	Zipf (0.75) e equação relacionando com o Zipf	OPT, CC, QC, LRU-2, LRU, LFU e FIFO	Simulador próprio sem código aberto
Bhat et al. (2018)	Zipf (1) e fixo o intervalo de chegada de 3 seg	-	SD-WANs
Koch et al. (2018)	Dados capturado do youtube	SLRU, ARC e ACDC	Simulador próprio com código aberto
Desmouceaux et al. (2019)	Zipf (0.1 a 0.9)e Zipf (0.1 a 0.9)	LRU filter e LRU	VPP
Pal e Kant (2017)	Zipf (0,6)	LRU, SMA, EWMA e AR	CCN routers
Wang e Chen (2018)	-	IFO, LRU e CLOCK,	CCN routers
Taher et al. (2018)	-	LFU, LRU e CCP	Analisou outro trabalho
Neves (2015)	Zipf (0.275) e Poisson (3)	CC, CARTE e RTBC	Simulador próprio sem código aberto

Fonte: Autoria própria

2.2 Algoritmos de substituição

2.2.1 RANDOM

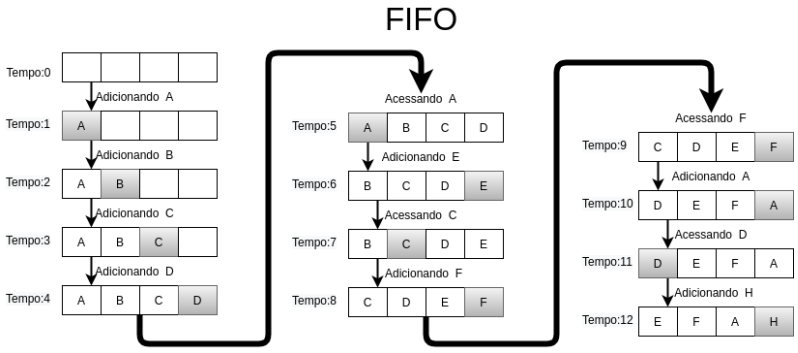
Segundo Gagne (2015), Random é um algoritmo aleatório que não usa nenhum método de escolha. Como não tem uma lógica para descarte, não tem como prever qual dado será descartado da cache, podendo ser qualquer dado em uma faixa que estende da classificação “muito frequentemente usado” até a classificação “raramente usado”. Ele utiliza poucos recursos do sistema e não apresenta ou possibilita um desempenho satisfatório em relação a LRU e LFU.

2.2.2 FIFO (First In First Out)

O autor Gagne (2015) descreve a lógica desse algoritmo como sendo equivalente ao funcionamento de uma fila, na qual o primeiro a chegar será o primeiro a sair. Essa estrutura em geral não é muito usada puramente, devido ao possível descarte de dados que são constantemente referenciados. Na Figura 3, é apresentado o funcionamento do FIFO, onde o algoritmo adiciona e remove os blocos de acordo com o funcionamento do algoritmo.

Na Figura 3, podemos ver que é adicionando os blocos na fila sem problema até o Tempo 4 da imagem. A partir desse momento, a memória está cheia, para adicionar outro elemento é necessário remover um elemento. É removido o primeiro elemento adicionado, pois esse algoritmo funciona como fila, assim o primeiro a entrar será o primeiro a sair. No Tempo 7 da Figura 3 é demonstrado outro detalhe desse algoritmos que é quando um bloco requerido já se encontra na memória, assim como demonstrado na figura, o bloco acessado permanece no mesmo lugar e com isso afirmamos que a consulta da informação que esteja na memória não influencia na fila.

Figura 3 – Exemplo usando FIFO em uma lista de 4 posições, associando um contador de tempo para demarcar o instante de entrada do conteúdo na cache.



Fonte: Autoria própria

2.2.3 LRU (Least Recently Used)

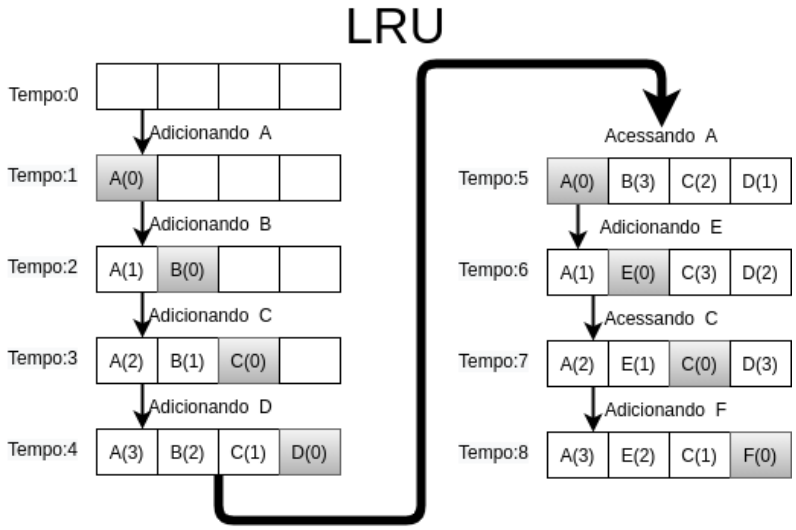
De acordo com Gagne (2015), o algoritmo LRU seleciona o elemento menos recentemente acessado para descartar. Ele foi desenvolvido pensando que o dado menos recentemente acessado não seria acessado novamente. Há diferentes modos de implementá-lo, sendo dois destes modos descritos no trabalho desse autor, os quais são também descritos a seguir.

A primeira maneira é associando o momento do último acesso a cada dado, e isso implica que será necessário atualizar sempre todos contadores a cada referência a um objeto, já que o objeto acessado terá seu contador zerado e os contadores dos demais objetos precisam ser incrementados. Os contadores são uma variável, cada bloco da memória tem uma variável e nessa variável tem um valor inteiro como demonstra na Figura 4 o valor que está dentro dos parênteses. Os contadores se modificam toda vez que um bloco na memória for acessado ou adicionado na memória, assim o valor da variável 0 (zero) representa o bloco adicionado ou acessado recentemente. Para descartar, deverá percorrer todos os dados para ver qual foi acessado menos recentemente, ou seja, aquele que possui um maior número no seu contador.

O segundo modo é através da lista encadeada, no qual os elementos estariam ordenados pelo último momento de acesso. Na Figura 4, é possível ver um exemplo de uma execução LRU em uma lista que tem 4 posições. Nos primeiros passos, adicionam-se os elementos que foram acessados, mas quando chega o tempo 5 de execução, a lista está cheia e, para adicionar mais elementos que serão acessados, é necessário adicionar na

mesma posição do elemento menos recentemente acessado. Outro aspecto que pode ser observado é que, a cada execução, são atualizados os valores que estão associados no momento de acesso.

Figura 4 – Exemplo usando LRU em uma lista de 4 posições, associando uma estampa de identificação da ordem mais recente de acesso para cada dado.



Fonte: Autoria própria

2.2.4 LFU (Least Frequently Used)

Conforme Gagne (2015), a lógica do algoritmo LFU seleciona os dados menos referenciados para descarte. Essa ideia é implementada com um contador em cada bloco da memória e o contador do bloco vai calcular mais um toda vez que for acessado. Quando necessária a substituição, o algoritmo verifica toda a lista e seleciona o elemento com valor do contador menor para excluí-lo.

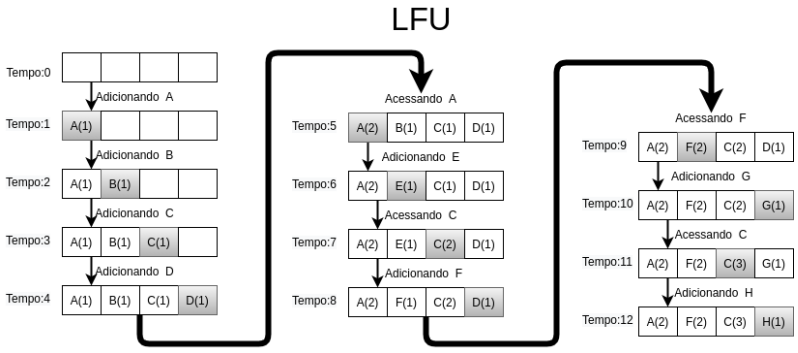
Desta forma, evita-se que as informações que são mais usadas sejam retiradas da cache, tendo em vista não considerar que um dado pode ser usado muito em um determinado momento, mas depois de um certo tempo ele passa a ser desnecessário. Além disso, um objeto, com valor muito elevado em seu contador, pode demorar muito para sair da cache ou talvez nem saia, apesar de possivelmente não receber futuros novos acessos. Desta forma, os novos objetos que têm dificuldade em competir com os demais que já estão há mais tempo na memória, tendem a ser mais frequentemente substituídos.

Por esse motivo, é implementado *reset* periódico sobre os contadores dos objetos armazenados para evitar que o valor do contador fique muito grande. A periodicidade

deste *reset* tem que ser bem ajustado para não causar valores contadores muito altos e nem muito baixos.

Na Figura 5, é possível ver um exemplo de uma execução LFU em uma lista que tem 4 posições. Nos primeiros passos, adicionam-se os elementos que foram acessados, mas quando chega o tempo 5 de execução, a lista está cheia e, para adicionar mais elementos que serão acessados, é necessário adicionar na mesma posição do elemento com menor valor contados . No entanto no tempo 5 ocorre acesso ao bloco A, aumentando o contador da frequência de acesso, no tempo 6 ocorre a adicção de um novo bloco é necessário remover um item da memoria e é removido o primeiro bloco com menor contador.

Figura 5 – Exemplo usando LFU em uma lista de 4 posições, associando um contador de tempo para cada dado.



Fonte: Autoria própria

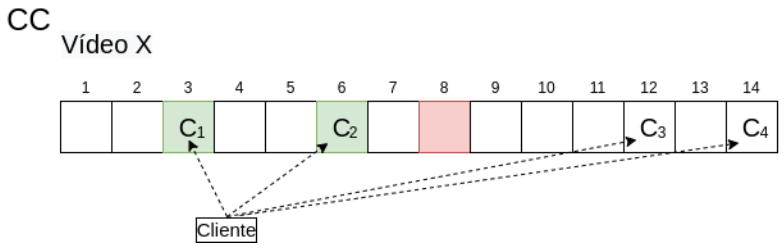
2.2.5 CC (chumk-based caching)

Hong, Vleeschauwer e Baccelli (2010) propuseram um algoritmo chamado de CC, o qual traz uma abordagem diferente dos algoritmos anteriores. O algoritmo verifica quantos clientes estão assistindo ao filme e quanto destes clientes ainda não passaram desde trecho do vídeo. Um termo pouco usual que descreve o tipo desse algoritmo é *look ahead*, pois ele analisa se os clientes podem necessitar desse trecho do vídeo.

Na Figura 6 temos um exemplo de um vídeo que chamamos de vídeo x e é repartido em uma estrutura de 14 pedaços iguais. O bloco em vermelho é o bloco que está na cache e para calcular a pontuação do bloco que está na cache, deve-se olhar quantos clientes ainda não passaram pelo bloco, são os dois blocos em verde. A pontuação do bloco será dois, pois há uma grande possibilidade de enquanto dois clientes que estão nos

blocos verdes cheguem no bloco que já está armazenado, enquanto dois outros clientes estão no fim do vídeo já passaram dessa parte, provavelmente não acessarão aquele bloco da cache.

Figura 6 – Exemplo Funcionamento do algoritmo CC com 3 filmes diferentes.



Fonte: Autoria própria

2.2.6 CARTE (*Current demAnd Rather Than futurE*)

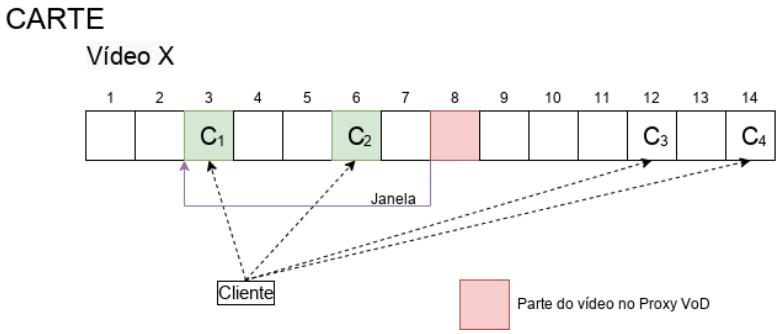
O algoritmo CARTE(*Current demAnd Rather Than futurE*) foi proposto por Neves (2015) e é um algoritmo que possui alguns aspetos semelhantes ao algoritmo CC, mas com alguns diferenciais para maximizar a produtividade do *proxy*. A principal estratégia adicionada nesse algoritmo em relação ao CC é a implementação de uma janela de tempo que limita o intervalo de análise para cálculo da prioridade de cacheamento em cada trecho de vídeo. Dessa formar, o algoritmo consegue calcular a densidade de clientes para esse intervalo e classificar a importância do trecho de vídeo.

Cabe salientar que a versão original do CARTE vincula prioridade de cacheamento as sequências de vídeos, tal que uma sequência de vídeo corresponde a um trecho de vídeo que separa dois clientes adjacentes que assistem a este mesmo vídeo. Diferentes sequências de vídeo, portanto, coexistem na memória do *proxy* que prioriza o armazenamento em sua memória das sequências com maior prioridade de cacheamento (maior densidade de clientes em um intervalo que precede cada uma destas sequências).

Neste trabalho, optou-se por desenvolver uma versão simplificada do CARTE para uso como referência comparativa nos testes de desempenho com o LRU e LFU com janela. O diferencial dessa versão simplificada em relação à versão original é aqui a versão simplificada vincula prioridades de cacheamento aos blocos de vídeo e não as sequências de vídeo como no algoritmo de referência. Ao contrário das sequências de vídeo que tem tamanho variável, com base no posicionamento dos clientes, os blocos de vídeo têm tamanho fixo, ou seja, todos os blocos têm o mesmo tamanho.

A Figura 7 apresenta um exemplo de como funciona para calcular a pontuação do bloco em rosa, o bloco rosa corresponde a uma parte do Video X que é armazenado na cache. Para calcular a pontuação do bloco rosa, é necessário o número de clientes que ainda não passaram desse trecho do filme e que estão dentro da janela de análise. O início da janela inicia no bloco anterior ao bloco rosa que é o bloco 7 e vai até o bloco 3, esse exemplo está trabalhando com o comprimento de janela com 5 blocos.

Figura 7 – Exemplo Funcionamento do algoritmo CARTE com 3 filmes diferentes.



Fonte: Autoria própria

2.3 Comportamento das requisições de vídeo

Segundo informações encontradas na literatura, o comportamento das requisições de vídeo recebidas por um servidor costuma ser descrito através de três distribuições principais:

- Poisson que é descrita por Almeida et al. (2001).
- Zipf que é descrita por Dan, Sitaram e Shahabuddin (1996).
- Mandelbrot-Zipf (MZipf) descrita por Hefeeda e Saleh (2008).

O trabalho de Almeida et al. (2001) tem como objetivo fornecer dados para gerar cargas de trabalho sintéticas. A distribuição de Poisson saiu da análise feita de *logs* de servidores de *streaming* e ela serve para descrever o intervalo entre as chegadas das requisições dos clientes no servidor. Em sua implementação, é usada uma função aleatória para gerar o intervalo entre entradas sucessivas de dois clientes. Esse intervalo converge para um valor médio em torno do coeficiente de Poisson escolhido.

A distribuição da popularidade dos vídeos foi modelada utilizando a lei de Zipf, no qual Dan, Sitaram e Shahabuddin (1996) descrevem a frequência dos elementos de

vídeo disponíveis em um conjunto (acervo). O coeficiente Zipf é a inclinação da linha do gráfico que descreve o comportamento da frequência para cada vídeo e, quanto maior o coeficiente, maior é a concentração das requisições sobre os vídeos mais populares do acervo. Sendo assim, quanto mais próximo de zero for o valor do coeficiente Zipf, mais equilibrado será o acesso aos vídeos.

A coluna 2 da tabela 4 apresenta as configurações utilizadas na literatura para as distribuições de Zipf e Poisson, com o objetivo de justificar as configurações adotadas nós simulador construído nesse trabalho. Outro fato importante é que a maioria dos autores usam valor alto para Zipf, o que sugere que o comportamento da rede funciona focando requisições em vídeo.

O trabalho de Lee et al. (2019) traz um debate sobre o uso de distribuição Zipf e MZipf, sendo possível entender que a distribuição MZipf é uma derivação do Zipf mais complexa. Essa discussão ocorreu devido à análise da distribuição da popularidade de vídeo que é medida pela BBC *(British Broad-casting Corporation) iPlayer*, o serviço de distribuição de vídeo mais popular do Reino Unido. A distribuição Mzip conseguiu replicar mais precisamente a distribuição da popularidade de vídeo medido pela BBC em comparação com a Zipf. Os autores usaram às duas distribuições em suas simulações e ao comparar os resultados finais obtiveram uma pequena variação, que não afetou a conclusão do trabalho.

3 VARIANTES PARA LRU E LFU

Este capítulo apresenta as estratégias utilizadas com intuito de se buscar o aprimoramento dos algoritmos clássicos LRU e LFU com detalhamento de cada uma. A seção 3.1 traz a base da implementação e referências da ideia. A seção 3.2 apresenta um relato da implementação sobre o LRU com janela. A seção 3.3 apresenta relato sobre o LFU com janela.

3.1 Proposta de implementação

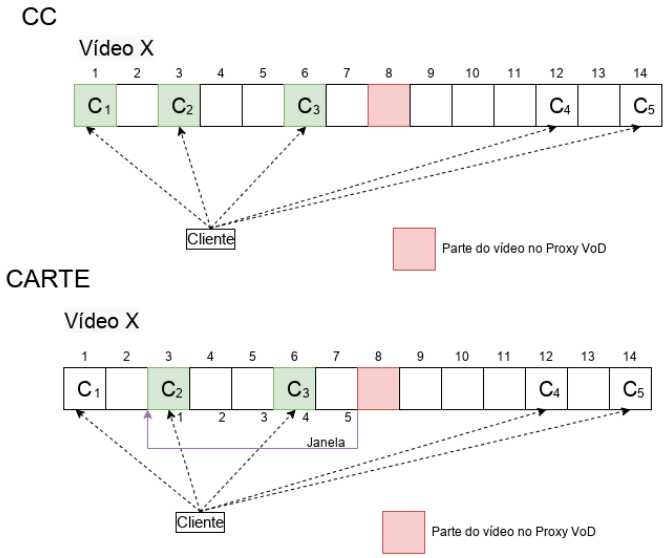
O algoritmo LRU tem diferentes modos de implementação são descritos no seção 2.2.3 e o algoritmo LFU não implementado normalmente sem o reset como descrito no seção 2.2.4. O algoritmo LRU tem um contador que tem que ser sempre atualizado ou tem que percorrer filas, esses modos de implementação são descritos no capítulo 2.4 LRU e o algoritmo LFU não implementado normalmente sem o reset como descrito no capítulo 2.5. Esses algoritmos trabalham com algumas estratégias para deixá-los com melhor rendimento. No livro do William (2010), é falado sobre o uso de variações dos algoritmos e locais onde podemos encontrar os algoritmos. O LRU ou variação dele pode ser encontrado na implementação de caches de alguns processadores como o Intel 80486 e nos algoritmos de substituição de página de memória, já o LFU pode ser encontrado somente nos algoritmos de substituicao de pagina de memória em kerneis de sistemas operacionais.

Ao analisar o algoritmo CC (HONG; VLEESCHAUWER; BACCELLI, 2010), percebe-se que os autores estabelecem a prioridade de descarte para cada bloco em memória (de um vídeo) com base na quantidade de clientes que estão assistindo a este vídeo, mas só são contabilizados os clientes que estejam assistindo esse vídeo e que não passaram do bloco que está sendo calculado a prioridade. Já no trabalho de Neves (2015), ele propõe o algoritmo CARTE, limitando a análise do trecho que será usado para levantar a quantidade de clientes, ou seja, no lugar de analisar todos os blocos anteriores ele analisa uma janela de tempo dos blocos, a janela de tempo é sempre iniciada no bloco anterior do bloco que será classificado.

A Figura 8 trás um exemplo para CC e CARTE, em que as linhas dos blocos representam um vídeo dividido em partes. Existem 5 clientes assistindo esses vídeos, todos os clientes andam da esquerda para a direita e o bloco vermelho representa o bloco

que está na cache. No entanto, para calcular a prioridade do bloco vermelho no algoritmo CC, basta contar o número de clientes que ainda não passaram pelo bloco vermelho, portanto 3 blocos verdes. Quanto ao algoritmo CARTE, basta contar o número de clientes que ainda não passaram no bloco vermelho e que estejam dentro da janela tempo que são os 2 blocos verdes.

Figura 8 – Exemplo do funcionamento dos algoritmos CC e CARTE. O algoritmos CARTE está usando uma janela de tempo de 5 blocos



Fonte: Autoria própria

A partir da análise desses dois trabalhos anteriores, surgiu a ideia de se aplicar estratégias semelhantes às observadas em CC e no CARTE, mas aplicá-las aos LRU e LFU, uma vez que esses dois algoritmos de substituição são importantes conforme se observa em estudos na literatura de VoD e seu uso é bem abrangente, logo, a utilização e modificação deles para melhorar seu desempenho em um *proxy* de VoD pode ser interessante. Um melhor detalhamento desta estratégia está especificado nas seções 3.2 e 3.3

3.2 Implementação do LRU com janela

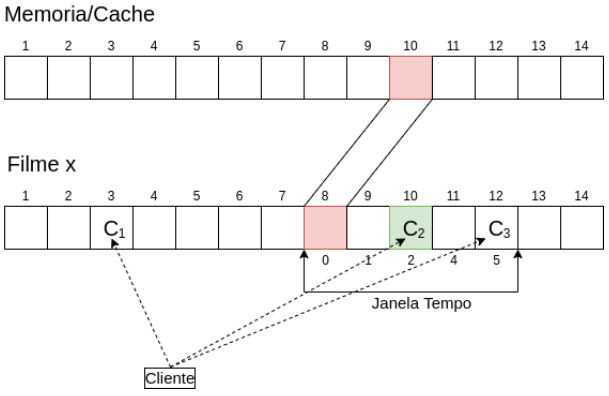
Baseado no algoritmo LRU, no qual o conteúdo acessado mais recentemente é o que fica na memória, foi desenvolvida uma estratégia em que é classificada a prioridade de cacheamento para um bloco de vídeo na cache com base no histórico de acesso do vídeo. Simplesmente é calculado a distância de bloco com o cliente mais próximo alocado dentro

de uma janela de tempo virtualmente posicionada após o bloco (isto é, em direção ao final do vídeo) no mesmo filme.

A Figura 9 apresenta um exemplo em que a memória do proxy dispõe de 14 posições para armazenamento de bloco e, especificamente, a posição 10 desta memória armazena atualmente o bloco de número 8 do filme X, bloco este para o qual se deseja calcular a prioridade de armazenamento. O filme X que estamos usando neste exemplo tem ao todo 14 blocos[1] e contém 3 clientes, C1, C2 e C3 que estão acessando, neste instante, respectivamente, os blocos de número 3, 10 e 12. A janela do tempo, utilizada pelo algoritmo LRU modificado, está sendo composta neste exemplo por 5 blocos, do 8 ao 12, onde o cliente mais próximo, C2, do bloco de número 8 do vídeo (cliente que realizou acesso mais recente a este bloco) está 2 casas para frente deste bloco.

A janela tempo de análise foi implementada no simulador da seguinte maneira: primeiro, utiliza-se a posição da memória que tem bloco de vídeo para o qual se quer calcular a prioridade de cacheamento (nível de prioridade para se manter o bloco na memória); depois, busca-se calcular a distância dos clientes mais próximos desse bloco que já acessaram esse bloco. Após isso, levando-se em consideração uma janela tempo de um tamanho pré-estabelecido, o simulador pega o valor da janela e o subtrai da distância até o cliente mais próximo ao bloco, e esse valor é usado para classificar o bloco, no momento em que precisará substituir alguns blocos para entrar um novo pedaço de filme. Dessa maneira, pode-se usar 0 (zero) para representar quando não há cliente dentro da janela de tempo e 5 representaria que o cliente que está sobre o bloco.

Figura 9 – Exemplo do funcionamento do LRU com janela



Fonte: Autoria própria

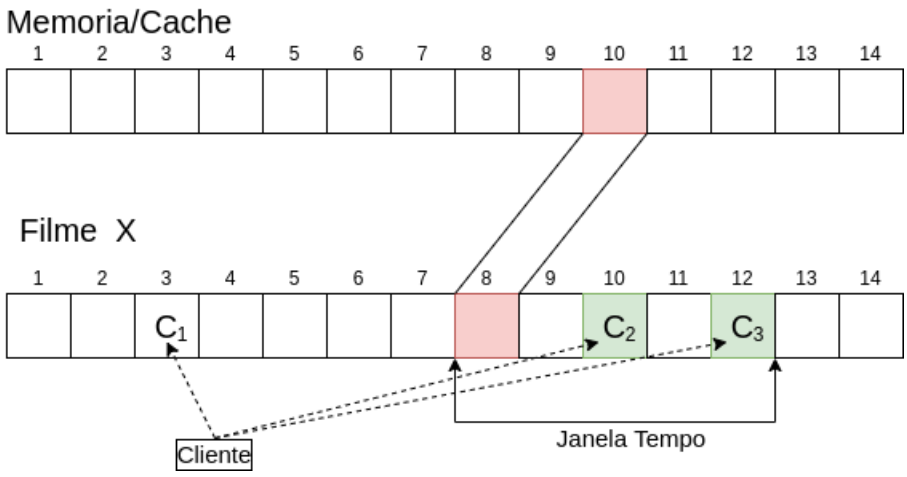
[1] Um filme normalmente tem tamanho 3,6GB, corresponde mais ou menos a 5852 blocos, sendo que cada bloco tem tamanho 0.63MB considerando um vídeo em formato HD

3.3 Implementação do LFU com janela

Com base no LFU, usando o princípio de quantas vezes um determinado conteúdo foi acessado, elaborou-se a seguinte tática para calcular a prioridade de cacheamento de cada bloco disponível na memória. É contabilizada a soma da quantidade de clientes que estão incluídos na janela de tempo e essa janela de tempo virtualmente posicionada após o bloco (isto é, em direção ao final do vídeo) no mesmo filme. Os macro passos do algoritmo seguem essa ordem: primeiro, se pega a posição da memória que tem bloco de vídeo; depois, se conta o número de clientes dentro da janela de tempo em cima do tempo do filme do bloco escolhido da memória; finalizado isso, se pega o valor contado, salva e o usa para classificar o bloco da memória, no momento em que precisar substituir alguns blocos para entrar um novo bloco de filme.

A Figura 10 apresenta um exemplo de funcionamento. Nesse exemplo, a memória tem 14 posições e é classificada a posição 10 da memória que contém armazenado o bloco número 8 do respectivo filme. O filme possui 14 blocos e contém 3 clientes dentro do filme que são C_1, C_2 e C_3 e estão em cima do último bloco que eles solicitaram. A janela do tempo representada no exemplo é composta por 5 blocos de vídeo, do 8° ao 12° bloco, e possui uma quantia de 2 clientes dentro deste espaço, sendo estes clientes C_2 e C_3. Usa-se esse valor 2 para representar a frequência de acesso, sendo que, quanto mais clientes tiverem dentro da janela de tempo, maior será a prioridade para preservação do bloco na cache.

Figura 10 – Exemplo do funcionamento do LFU com janela



Fonte: Autoria própria

4 AMBIENTE EXPERIMENTAL

Este capítulo abordará as seguintes seções: a seção 4.1 apresentará a justificativa para a construção de um simulador; na seção 4.2 serão abordados os componentes do simulador e seu funcionamento; na seção 4.3 serão apresentados os parâmetros de configuração do sistema; e, por último, na seção 4.4, serão apresentados os testes e os resultados produzidos a fim de validar o simulador.

4.1 Aspectos que levaram a construção de um novo simulador

Na literatura, encontram-se trabalhos de autores que desenvolveram simuladores semelhantes ao elaborado para o presente trabalho, tais como o trabalho de Neves (2015), Osman e Osman (2018) e Koch et al. (2018). No entanto, analisando mais detalhadamente esses trabalhos, percebe-se que seus simuladores incluem um conjunto de recursos adicionais que não são avaliados neste momento, pois podem causar algumas dificuldades na execução e desenvolvimento das novas propostas de algoritmos de substituição.

Sendo assim, neste trabalho, o foco será dado na avaliação da taxa de acertos dos algoritmos implementados até aqui, uma vez que a proposta para aprimoramento destes algoritmos visa o aumento da eficiência do *proxy* VoD. Devido a isso, optou-se pelo desenvolvimento de um simulador próprio com ênfase em modelar minimamente os aspectos essenciais de funcionamento do *proxy*, obtendo-se assim, de forma ágil (através de simulações objetivas com curto tempo de resposta), os resultados para a métrica de desempenho pretendida.

4.2 VoDProS -*Video on demand proxy Simulator* [1]

Um simulador em Python, que é uma linguagem interpretada, foi desenvolvido com o intuito de contemplar os componentes necessários para simular um servidor VoD, tais como uma fila de clientes, uma matriz de classificação, uma árvore binária, uma memória cache, uma fila de espera, uma fila de substituição e tempo. Estes componentes estão descritos nas seções a seguir.

[1] Código-fonte disponível: https://github.com/Koch1/VoDProS-Video-on-demand-proxy-Simulator

4.2.1 Fila Clientes

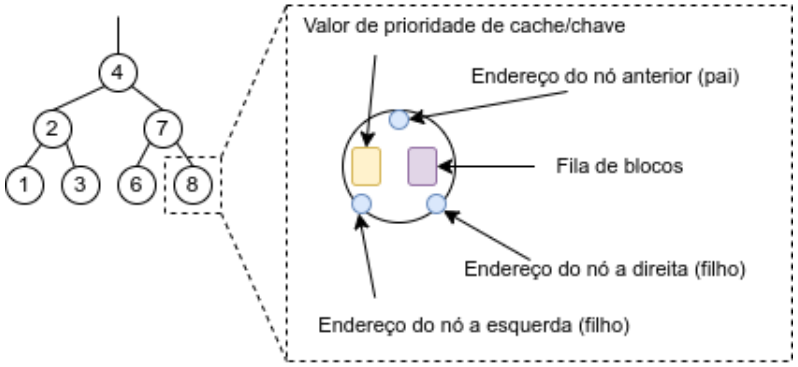
A fila de clientes armazena os clientes que precisam ser atendidos até que esses clientes terminem suas sessões de vídeo. Para isso, é lido um arquivo em que cada linha armazena informações sobre a requisição de vídeo de um cliente e em cada uma dessas linhas há 4 colunas: a primeira contém o id do cliente; a segunda a identificação do filme; a terceira o tempo em segundos que o cliente solicita o vídeo; e, na quarta e última coluna, os períodos de tempo que o cliente está assistindo ao vídeo. Os detalhes de funcionamento podem ser vistos na Figura 12, onde é possível entender a execução do simulador para adição de novos clientes.

4.2.2 Matriz de classificação e árvore binária

A classificação de blocos é feita por meio da matriz de classificação que é alimentada a cada nova solicitação do cliente. Seu tamanho varia de acordo com o número de diferentes blocos de vídeo armazenados pelo *proxy*, ou seja, a cada bloco que entra no cache deve ocupar uma posição neste *array*, pois é onde está armazenado o valor da prioridade do cache do bloco. Quando o bloco é substituído, seu valor de classificação é removido da árvore binária.

Uma árvore binária é a estrutura de dados usada para organizar os dados da matriz de classificação, árvore binária usada está exemplificada na Figura 11. Cada nó na árvore de chaves corresponde a um valor de prioridade de cache e, dentro do nó, há uma fila de blocos que têm a mesma valor de prioridade de cache, endereço de nó da esquerda, endereço de nó direita e endereço de nó anterior. O intervalo é um valor cumulativo em que você acessa o valor do intervalo anterior e adiciona mais 1 ou subtrai mais 1 nos blocos de requisitos apropriados. Quando ocorre uma alteração de classificação na lista, a mesma operação é executada na árvore binária quando os blocos reordenados estão na memória.

Figura 11 – Árvore binaria e componentes do nó



Fonte: Autoria própria

4.2.3 Memória cache

Nesse componente do *proxy* simulado, é feito o armazenamento das informações sobre os blocos de vídeos, cada elemento contém as seguintes informações: id do filme, id do bloco e a quantidade de clientes acessando o bloco. A memória da cache simulada corresponde a uma matriz na qual o número de elementos por linha representa a quantidade de blocos de vídeo armazenados para cada vídeo. Os *frames* dos vídeos em si não são armazenados por questão de espaço, para que seja possível simular cenários de funcionamento de *proxies* VoD com uma memória maior do que a do computador usado para a simulação.

4.2.4 Fila de espera

Na fila de espera, são adicionados os pedidos de blocos que não estão em cache para que seja possível fazer o pedido destes blocos ao servidor principal. Ao fim da leitura da fila de clientes, organiza-se a fila de espera usando três critérios: primeiro, o maior número de requisições repetidas para o mesmo filme e bloco; segundo, o maior número de requisições para o mesmo filme; e, por último, a ordem das requisições feitas. Essa organização da fila visa atender o maior número possível de pedidos não repetidos,que podem ser atendidos dentro do limite de banda disponível (entre o *proxy* e o servidor principal).

4.2.5 Fila de substituição

Essa fila é criada depois que a "fila de espera" terminar de organizar e escolher as solicitações que serão buscadas. A fila de substituição contém blocos como os seguintes dados: as posições da memória em que o vídeo está localizado; idFilme; idBloco;. Ela é gerada através de uma leitura in-ordem da árvore binária, a leitura traz uma fila das junções das fila de blocos de dentro do nó. Essa fila retornada tem o tamanho da "fila de espera" dentro do limite de banda disponível (entre o *proxy* e o servidor principal). Só há dois critérios para classificação de blocos para substituição. O primeiro referente ao valor da "matriz de classificação" e o segundo é a ordem da fila de blocos com mesma classificação.

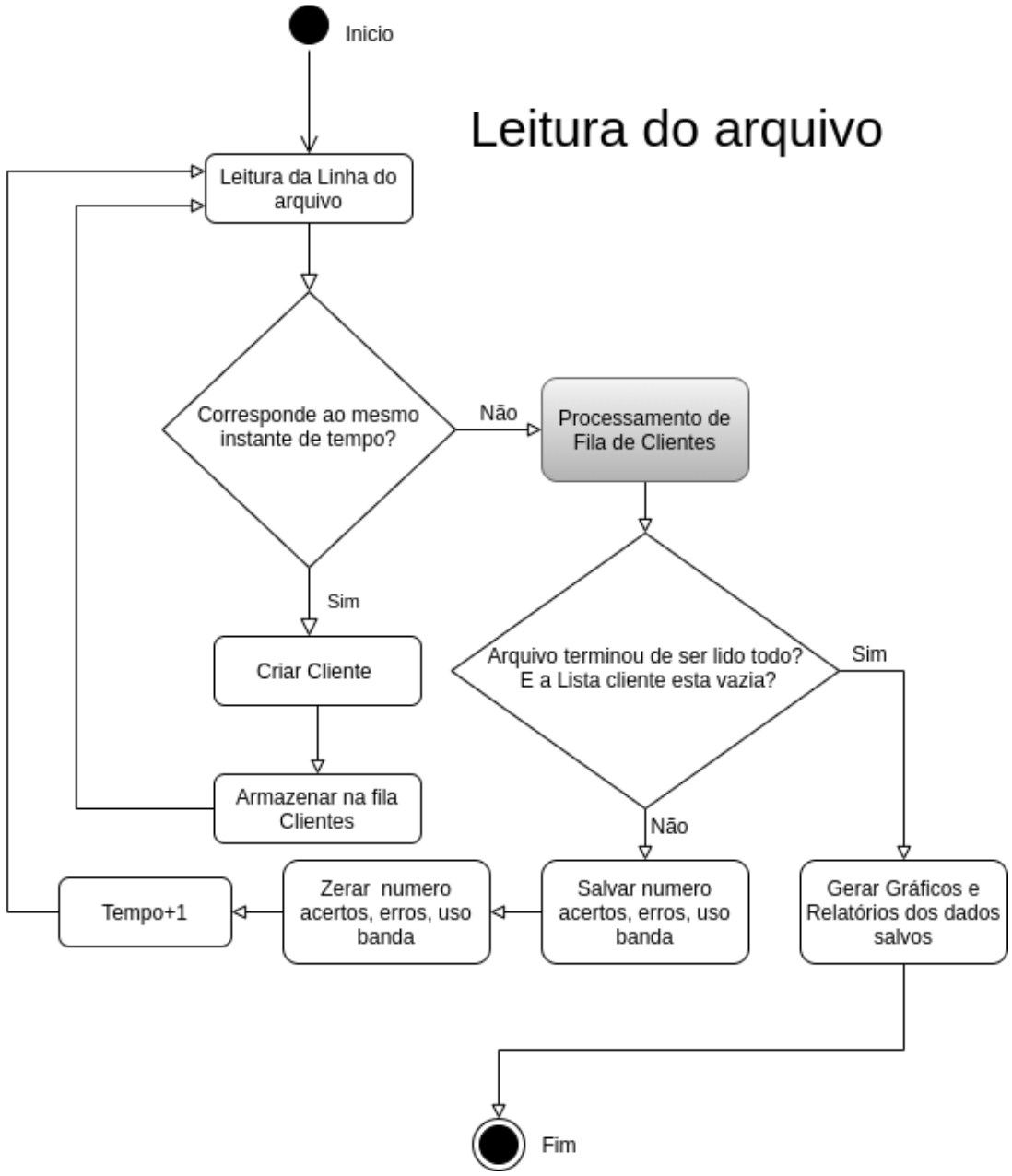
4.2.6 Diagramas UML para Descrição do Funcionamento do Simulador Implementado

As figuras 12, 13, 14 e 15 abaixo descrevem o funcionamento do simulador atual com detalhamentos. Esse *Unified Modeling Language* (UML) permite entender o funcionamento de cada simulação e poderá ser um guia para fazer um teste de mesa dela.

O tempo dentro da simulação é diferente do tempo usado para executar o simulador no computador hospedeiro. O tempo fora da simulação depende do algoritmo de substituição que foi utilizado e das configurações do computador usado. Esse tempo fora da simulação poderá variar muito, já o tempo na simulação sempre será igual independente dos parâmetros físicos e lógicos inerentes a infra-estrutura real de execução de um *proxy* VoD.

Dentro da simulação, cada segundo virtual equivale ao tempo necessário para realização das seguintes atividades pelo *proxy*: responder a todos os clientes ativos na simulação, criar e consumir a fila de espera e a fila de substituição e receber todas as solicitações feitas para o servidor principal.

Figura 12 – Diagrama de atividades UML sobre o funcionamento da leitura do arquivo no simulador.

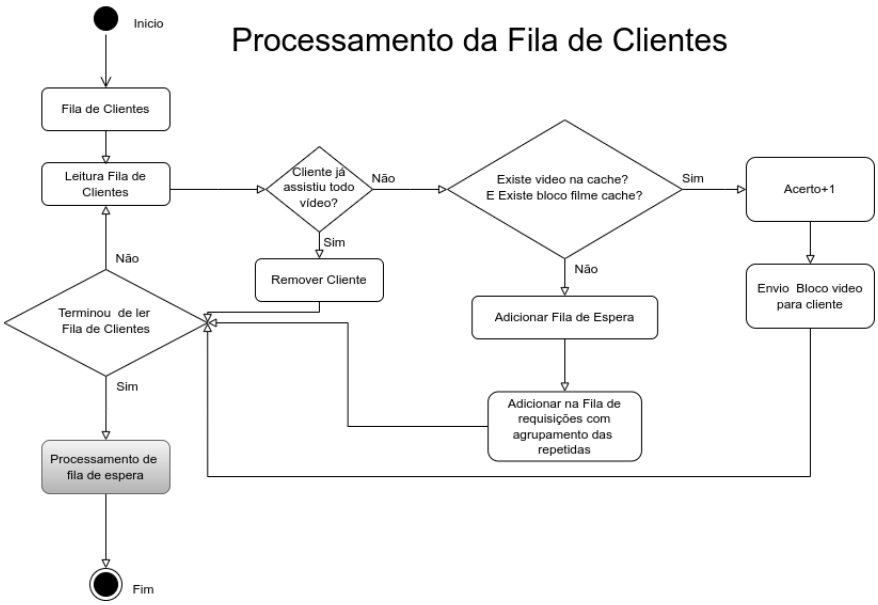


Fonte: Autoria própria

A Figura 12 traz o ciclo completo do funcionamento do *proxy*, primeiramente é realizado a leitura das requisições que estão no mesmo ciclo de tempo. Após a leitura, a fila de cliente é processada, os clientes que estão nesta fila são os clientes que estão assistindo os filmes, então eles precisam ser atendidos a cada ciclo até o final do filme. Antes de acabar o ciclo, é necessário verificar se terminou a leitura do arquivo de requisições e ver se tem cliente na fila de clientes. Caso tenha cliente para suprir é

necessário armazenar a taxa de acerto, erro e uso de banda e depois zerar os contadores de taxa de acerto, erro e uso de banda para próximo ciclo. Depois de verificado se não tem clientes para suprir, são gerados informações da simulação.

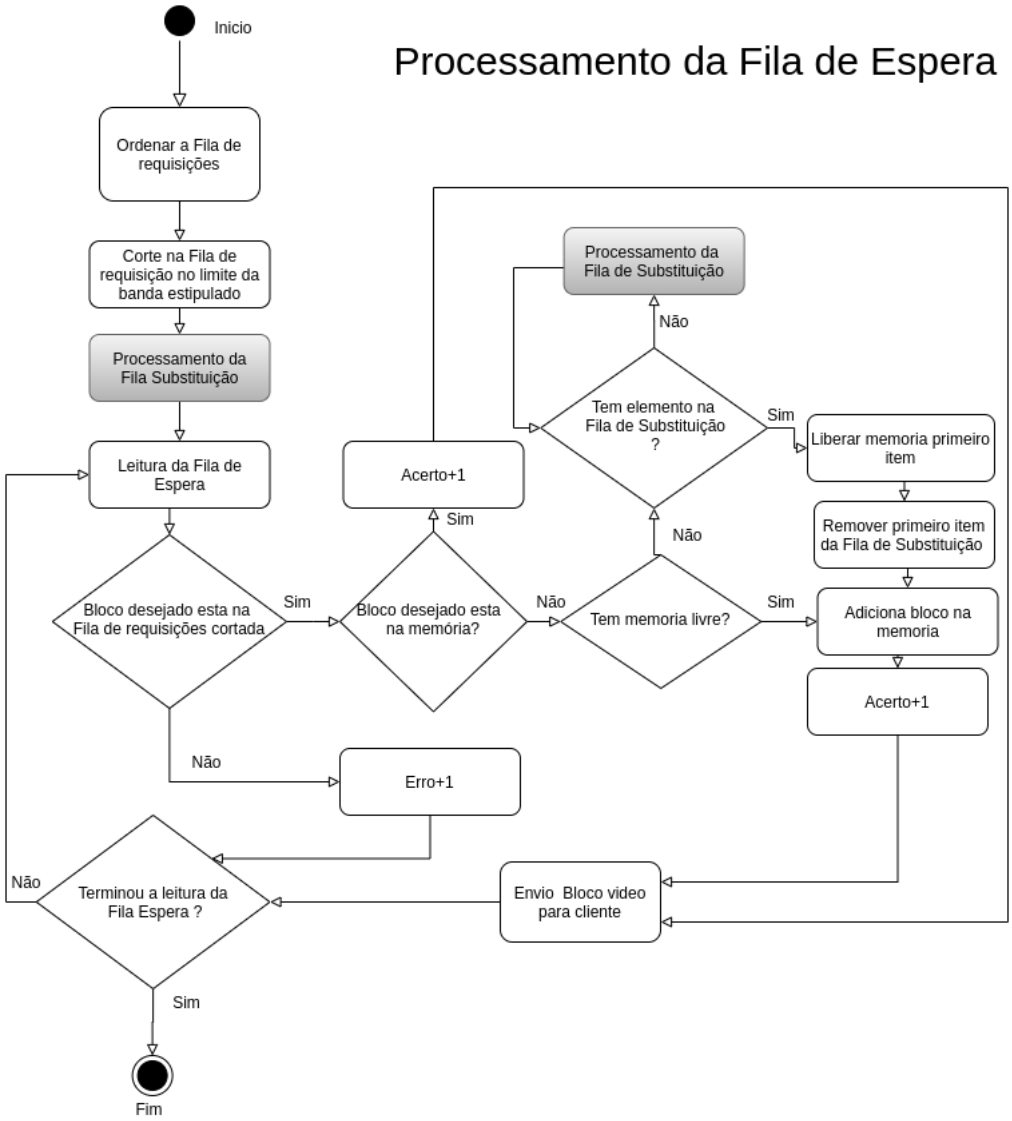
Figura 13 – Diagrama de atividades UML sobre o funcionamento do processamento da fila de Clientes no simulador.



Fonte: Autoria própria

Na Figura 13, temos os procedimentos que são feitos dentro do processamento da fila do clientes. Primeiro passo é a leitura da fila, na qual se pega o primeiro cliente da fila e verifica se ele já assistiu o filme inteiro, se sim o cliente é removido e então é passado para a próxima fila. Caso o cliente não tenha terminado, é procurado na cache se existe o bloco do filme solicitado, caso sim, é enviado o bloco de filme para cliente, e, caso não, o cliente entra em outra fila de espera.

Figura 14 – Diagrama de atividades UML sobre o funcionamento da processamento da fila de espera no simulador.

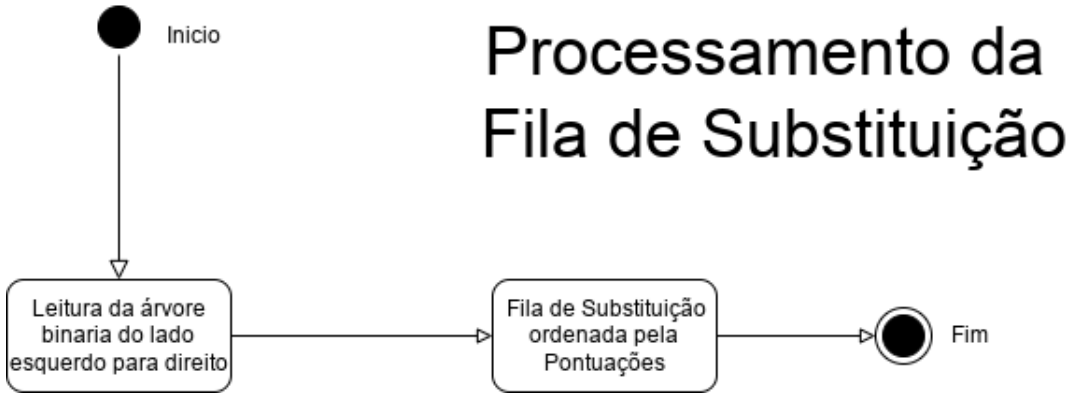


Fonte: Autoria própria

A Figura 14 traz o processamento da fila de espera que é constituída por todos os clientes que não conseguiram encontrar os blocos que necessitam na cache, então esses clientes terão que disputar o uso da banda *proxy* com o servidor principal. Primeira passo é agrupar as requisições repetidas e então gerar a fila de espera pela ordem de requisições que tem maior número de cliente solicitando. Após a fila gerada, será feito corte nesta fila, pois existe um limite de largura de banda do servidor *proxy*. Depois de buscar os blocos dos filmes que foram requisitados pela fila de espera, os blocos são distribuídos para os clientes, antes disso precisamos armazená-los na cache do *proxy*. Para ocorrer o armazenamento dos blocos, é necessário verificar se a cache está cheia, caso esteja cheia,

o novo bloco será salvo sobre o outro bloco existente, esse local de salvamento é indicado pela fila de substituição.

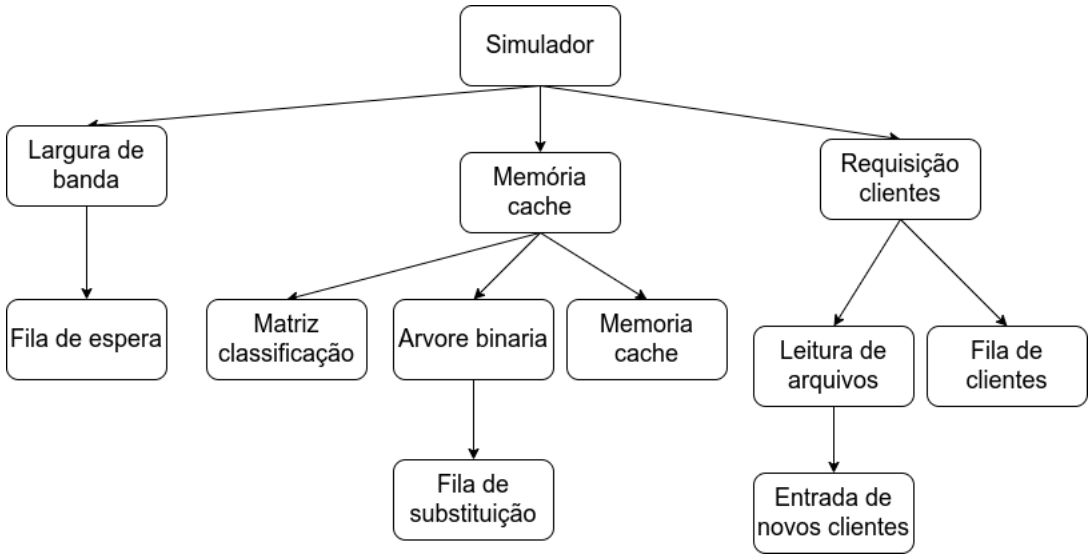
Figura 15 – Diagrama de atividades UML sobre o funcionamento do processamento da fila substituição no simulador.



Fonte: Autoria própria

A Figura 15 traz o processamento da fila de substituição que é gerada através da leitura da árvore binária do lado mais esquerdo para direito, assim temos uma lista com a pontuação organizada. A adição na árvore e recalculagem da pontuação na árvore é realizada a cada requisição dos clientes, essa pontuação varia de acordo com os algoritmos de substituição escolhido.

Figura 16 – Diagrama de componentes que descrevendo os elementos constituintes do simulador.



Fonte: Autoria própria

Na Figura 16 trás os diagramas de componentes, demostrando os principais elementos e os locais que eles fazem parte. Os principais componentes são a largura de banda, memoria cache e requisição de clientes, e esse componentes podem ter modificados as configurações.

4.3 Parâmetros de configurações do sistema

De modo a conseguir chegar a um conjunto de valores para configuração do simulador, foram revisados alguns trabalhos e decidiu-se tomar como base o trabalho de Neves (2015). O trabalho de Neves (2015) abrange todos os parâmetros que serão necessários e o tempo de execução da simulação é menor se comparado ao trabalho de Koch et al. (2018), Osman e Osman (2018).

As simulações foram executadas em 4 computadores com diversas configurações de hardware e software diferentes, no entanto, essa mudança de ambiente de execução não interfere no resultado das simulações, só afeta o tempo que cada simulação pode demorar para executar. Para todas as simulações foi definido uma padronização dos parâmetros, a qual está descrita nas Tabela 5 e Tabela 6.

Os parâmetros descritos na Tabela 5 são utilizados para gerar os arquivos de carga para a simulação e esses parâmetros são o coeficiente Zipf, coeficiente de Poisson, o número de vídeos, o tempo de simulação e o número de clientes.

O coeficiente Zipf define o comportamento da frequência para cada vídeo, a mudança de valor implica a quantidade de vezes que o mesmo vídeo é visualizado e quanto mais um vídeo específico for solicitado, maior a probabilidade de o vídeo estar no cache.

O coeficiente de Poisson encarregado pelo comportamento do intervalo de chegada dos clientes, a alteração de tal número pode influenciar na quantidade de clientes atendidos, pois temos recursos limitados para cada segundo da simulação.

O número de vídeos corresponde a quantidade de filmes que temos no acervo e uma mudança desse valor pode implicar a chance de ter a cache com conteúdo diversificado, mas depende muito do coeficiente Zipf para indicar a frequência dos vídeos.

O tempo de simulação (segundos) representa o tempo que foi simulado pelo simulador.

O número de clientes é a quantidade total de clientes que a simulação tem e esse valor é calculado pegando o valor do tempo de simulação e dividido pelo coeficiente de

Poisson, e o resultado gerado é arredondado para cima.

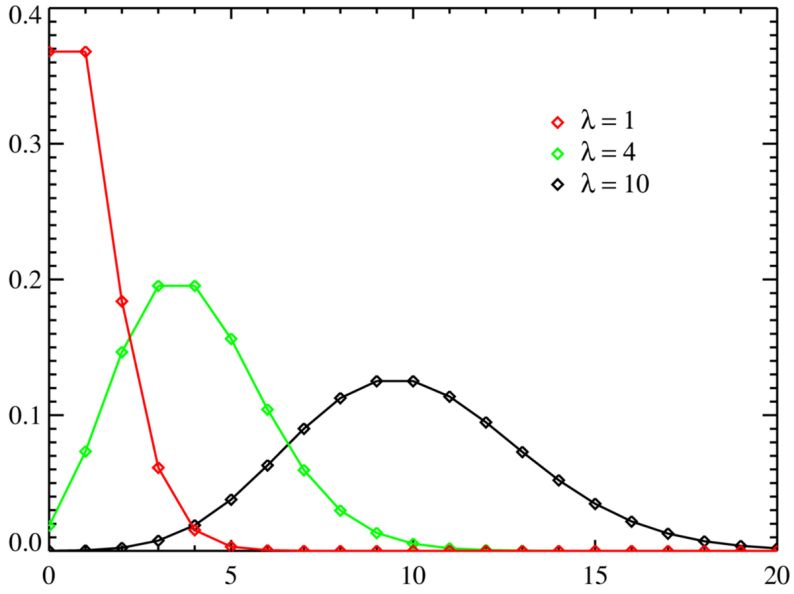
Tabela 5 – Configurações padrão para o arquivo que descreve as requisições dos clientes

Parâmetros	Valor
Coeficiente Zipf- (inclinação)	0,271
Coeficiente de Poisson- (seg.)	3
Número de vídeos	100
Tempo de Simulação (segundos)	7200 (2 horas)
Número de Clientes	2400

Fonte: Autoria própria

O tempo de simulação é aproximado, pois foi utilizada a distribuição de Poisson, que utiliza uma função aleatória para gerar o intervalo entre entradas sucessivas dos clientes. Esse intervalo converge para um valor médio em torno do coeficiente de Poisson escolhido, este intervalo escolhido é o pico da onda gerado pela distribuição de Poisson. A Figura 17 o eixo x representa o tempo de entrada e eixo y representa o número clientes, os clientes estão distribuídos pela área do gráfico, então observa-se que o gráfico há clientes com intervalos de chegada maiores ou menores.

Figura 17 – Função de probabilidade da distribuição de Poisson para vários coeficientes

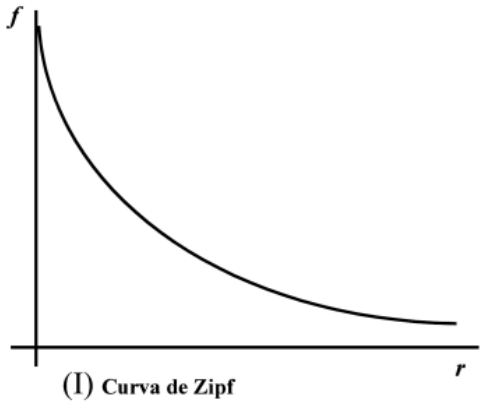


Fonte: https://pt.wikipedia.org/wiki/Distribui%C3%A7%C3%A3o_de_Poisson

A distribuição da popularidade dos vídeos utiliza a lei de Zipf, na qual Dan, Sitaram e Shahabuddin (1996) descrevem a frequência dos elementos de vídeo disponíveis em um conjunto (acervo). O coeficiente Zipf é a inclinação da linha do gráfico que

descreve o comportamento da frequência para cada vídeo e quanto maior o coeficiente, maior é a concentração das requisições sobre os vídeos mais populares do acervo, com um declínio exponencial como apresentado na Figura 18. Sendo assim, quanto mais próximo de zero for o valor do coeficiente Zipf, mais equilibrado será o acesso aos vídeos.

Figura 18 – Gráfico do comportamento da distribuição de Zipf



Fonte: (SOARES et al., 2008)

Neste trabalho, considera-se que o valor de 0,63 Megabytes corresponde ao tamanho de um bloco de vídeo contendo um segundo de vídeo, esse valor de 0,63 MB equivale a aproximadamente um segundo de um vídeo em HD. Usamos este valor para converter os parâmetros dentro do simulador em parâmetros reais conforme descrito no trabalho de Neves (2015), os valores resultantes dessas conversões de parâmetros são dados na Tabela 6.

No trabalho de Osman e Osman (2018), é comentado que se deve cuidar o aumento em demasia do tamanho da memória cache, pois pode não mostrar uma melhora significativa na taxa de acerto de todos os algoritmos. Com isso, devemos preparar e fazer testes para várias configurações de memória.

Tabela 6 – Configuração do simulador desenvolvido

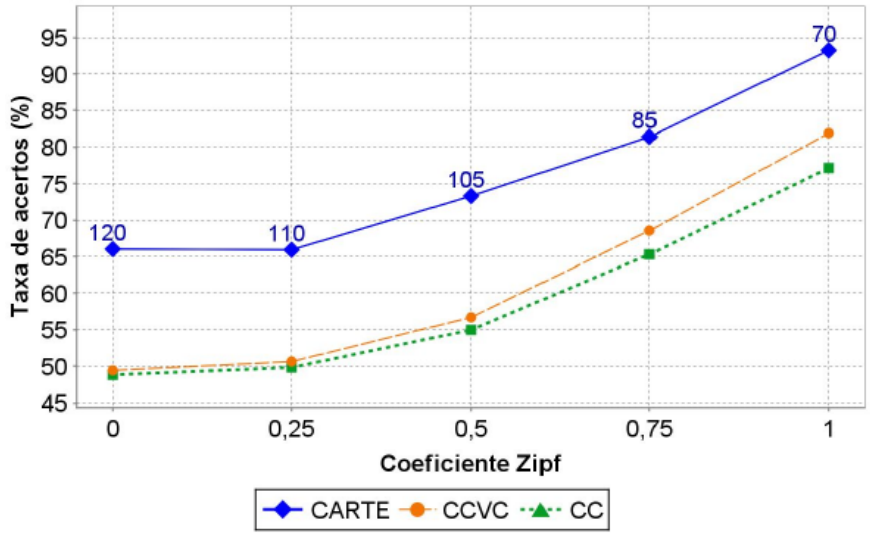
Duração dos vídeos	3.4 GB((90 min (5527 Blocos))
Tamanho da memória	14 GB(22756 Blocos)
Máxima Largura de banda no enlace servidor-*proxy*	3.4Gb/s (691 Blocos por segundo)

Fonte: Autoria própria

4.4 Validação

A validação dos resultados produzidos pelo simulador atual é feita através da comparação dos resultados obtidos para as implementações dos algoritmos LRU e LFU, considerando a utilização das configurações para funcionamento do simulador descrito nas Tabelas 5 e 6, mas alteramos a configuração do Zipf para 1. Para o fluxo de validação foram gerados 24 arquivos de carga usando as mesmas configurações para cada arquivo (baseadas na Tabela 5, mas alteramos a configuração do Zipf para 1). Comparando o resultado médio do algoritmo CC que está na Tabela 7 com valor CC da Figura 19, temos uma taxa de acertos muito próximos entre os dois, entretanto é uma diferença entre valor que poder ser compensado pelo desvio padrão ou variação dos valores.

Figura 19 – Gráfico sobre o tempo de serviço para a variação do coeficiente Zipf



Fonte: Autoria Neves (2015)

A tabela 7 demonstra resultados com execução de 25 simulações com valores padrões das Tabelas 5 e 6, mas com a configuração do Zipf igual a 1.0. Na parte inferior da Tabela 7 temos a média, desvio padrão e variação entre o menor e o maior de cada algoritmo de substituição. O valor do desvio padrão e da variação torna-se muito importante para comparação com outros trabalhos e para replicação de resultados. Serão usados para ajudar na comparação dos resultados das simulações que serão encontradas posteriormente, para verificar se as flutuações nos resultados podem alterar as classificações dos algoritmos de substituição.

Tabela 7 – Resultados da taxa de acertos e médias em relação aos algoritmos Random, LRU, LFU, FIFO e CC usando os valores padrão da Tabelas 5 e 6 mas com a configuração do Zipf igual a 1.0

Cenários	Random	LRU	LFU	FiFO	CC
0	59,72%	65,87%	52,60%	63,56%	70,70%
1	61,13%	67,11%	54,37%	65,20%	74,09%
2	60,20%	66,04%	53,59%	63,50%	72,05%
3	61,54%	67,04%	56,90%	65,98%	73,37%
4	61,92%	68,29%	54,34%	65,40%	74,64%
5	61,89%	67,77%	54,72%	65,25%	75,34%
6	61,12%	66,96%	55,28%	65,01%	74,72%
7	60,85%	67,44%	56,26%	65,04%	72,83%
8	61,52%	67,62%	54,91%	65,12%	74,17%
9	59,59%	64,34%	52,27%	62,56%	71,77%
10	59,64%	65,77%	53,13%	63,64%	72,03%
11	61,01%	66,61%	56,36%	64,54%	73,44%
12	60,62%	67,22%	55,56%	65,43%	71,85%
13	61,92%	68,90%	52,37%	65,47%	75,46%
14	61,70%	67,60%	55,58%	65,87%	74,24%
15	61,50%	67,17%	54,39%	64,95%	75,10%
16	60,24%	66,20%	54,32%	64,72%	72,55%
17	60,36%	65,04%	56,58%	63,16%	72,82%
18	60,65%	66,59%	52,61%	64,35%	72,83%
19	59,94%	65,64%	53,26%	63,81%	71,61%
20	61,45%	67,09%	57,31%	65,35%	74,32%
21	61,15%	67,48%	54,02%	64,65%	72,61%
22	61,64%	67,88%	55,74%	65,53%	74,35%
23	61,21%	67,51%	57,57%	65,51%	73,99%
24	61,95%	68,25%	56,53%	65,29%	75,23%
Media	60,98%	66,94%	54,82%	64,76%	73,44%
Desvio padrão	0,76%	1,06%	1,60%	0,90%	1,34%
Variação entre menor e maior valor	2,36%	4,56%	5,30%	3,42%	4,76%

Fonte: Autoria própria

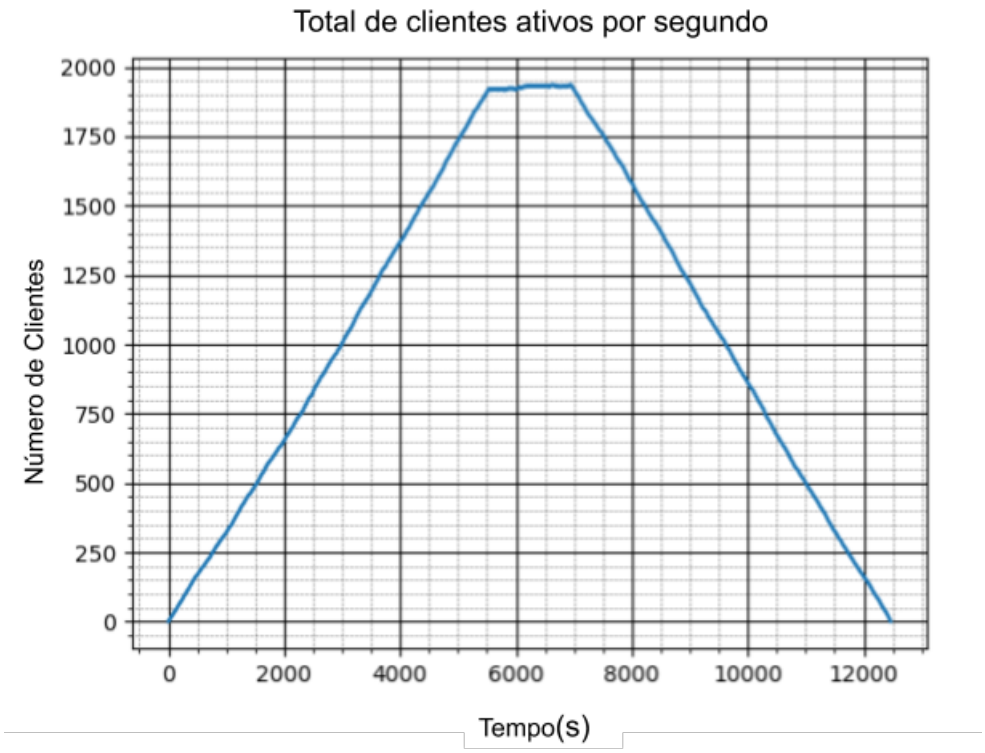
5 RESULTADOS

Este capítulo está dividido em três partes: a primeira parte (seção 5.1) trata de como foram calculados os resultados; a segunda (seção 5.2) parte aborda os resultados do LRU com janela; e a última parte (seção 5.3) trata dos resultados obtidos para o algoritmo LFU com janela.

5.1 Cálculo dos resultados e variação dos Parâmetros

O comportamento dos clientes dentro da simulação pode ser visto na Figura 20. Nesse gráfico, podem ser notados três momentos diferentes da simulação: a primeira parte tem reta crescente do 0 aos 5500 segundos, que corresponde ao fluxo de clientes que entram no *proxy*; a segunda parte, representada pelo intervalo de 5500 aos 6950 segundos contém o intervalo em que ocorrem tanto a entrada quanto a saída do cliente no *proxy*; e a terceira parte do 5950 aos 12500 segundos, contendo somente o fluxo da saída dos clientes.

Figura 20 – Gráfico total de clientes ativos por segundos de execução



Fonte: Autoria própria

Avaliaremos o intervalo de 5600 a 6600 segundos que corresponde ao momento em que existem a entrada e saída de clientes, e esse período corresponde ao comportamento mais frequente encontrado na vida útil do *proxy* VoD. Esse intervalo de 1000 segundos corresponde à amostra utilizada para calcular a média dos resultados, que é a soma da taxa de acertos dividida pelo número de amostras. Apesar de ser uma amostra pequena, traz os resultados necessários já que o cenário mantém um padrão de comportamento.

O período de inicialização do intervalo 0 a 5599 segundos e o período de desligamento do intervalo 6601 a 12500 segundos podem ser desprezados, pois afetam minimamente a taxa de acertos em servidores *proxy*, uma vez que servidores *proxy* VoD são equipamentos feitos para trabalharem vários dias, semanas, meses e anos sem a necessidade de desligar. No entanto, o período de inicialização e desligamento em nossa simulação pode influenciar na taxa média de acertos, já que estamos analisando uma pequena amostra que corresponde a 1000 segundos (16 minutos). Então, para obter valores da taxa de acerto mais precisos, optamos em calcular apenas a taxa média de acertos no intervalo de 5600 a 6600 segundos, para todas as simulações realizadas.

A taxa de acertos está sendo calculada pelo número de blocos entregues para os clientes divididos pelo número de requisição de clientes ativos e multiplicando por 100 a resposta. Os pacotes entregues são compostos pelos blocos encontrados na memória do *proxy* somados aos blocos que puderam ser transferidos do servidor para o *proxy* para suprir a falta de blocos no *proxy* (assumindo que estes pacotes também puderam ser repassados aos clientes, uma vez que o simulador atualmente assume que o *proxy* não limita o número de conexões de clientes).

Foram vários grupos de simulações, onde em cada grupo se buscou variar apenas um dos parâmetros de configuração do sistema, conforme descrito nas Tabela 8 e Tabela 9. Todos os algoritmos de substituição utilizam como entrada os mesmos arquivos de carga criados para cada grupo de simulação. Isso permite uma comparação mais justa, já que o coeficiente Zipf e coeficiente de Poisson têm variância como mostrado na Tabela 7. Os arquivos gerados usaram a Tabela 8 como base para as configuração padrão (valores fixos usados para os demais parâmetros durante o experimento de variação de um único parâmetro pontual). A faixa de variação determina os limites de variação dos parâmetros e Passo de Incremento é o valor que é incrementado para fazer a variação do parâmetro.

Optamos por fazer 30 simulações iguais com arquivos de cargas diferentes e depois calcular as médias de cada ponto gráfico, pois traz um resultado mais confiável

e consequentemente multiplica 30 vezes o tempo de processamento necessário. Apesar de termos calculado a média, devemos lembrar que existem os valores de desvio padrão e variação que estão na tabela 7.

Tabela 8 – Configurações do arquivo que contém clientes e requisições

Parâmetros	*Valor Padrão*	*Faixa de Variação*	*Passo de Incremento*
Coeficiente Zipf- (inclinação)	0,271	0-1	0.1
Coeficiente de Poisson- (seg.)	3	1-10	1
Número de vídeos	100	-	-
Tempo de Simulação (segundos)	7200(2 horas)	-	-
Número de Clientes	2400	-	-

Fonte: Autoria própria

A Tabela 9 traz as configurações feitas diretamente no simulador para todos os algoritmos, mas os algoritmos com janela requerem um parâmetro da janela que não possui um padrão. Os algoritmos que têm janela, tiveram uma variação de janelas personalizada, pois os algoritmos LFU com janela, LRU com Janela e CARTE trabalham com valores de janela diferentes. Através de simulações verificamos a faixa de variação e passo de incremento dos algoritmos com janela, para LFU com janela foi constatado faixa de varição de 50 a 3000 blocos (31,5 MB a 1,84 GB) com Incremento 50 blocos (31,5 MB), já o CARTE tem a faixa de varição de 10 a 300 blocos (6,3 MB a 189 MB) com Incremento 10 blocos (6,3 MB) e o LRU com Janela tem a faixa de varição de 0 a 50 blocos (31,5 MB) com Incremento 10 blocos (6,3 MB)

Tabela 9 – Configuração do simulador desenvolvido

Parâmetros	*Valor Padrão*	*Faixa de Variação*	*Passo de Incremento*
Duração dos vídeos	3,4 GB	-	-
Tamanho da memória	14GB	1-25	1
Máxima Largura de banda no enlace servidor-*proxy*	3,4Gb/s	0.5Gb/s-4Gb/s	0.4Gb/s

Fonte: Autoria própria

5.2 Resultados de LRU versus LRU com janela

Ao comparar os resultados do LRU e LRU com a janela, foi registrada piora de desempenho significativa na média da taxa de acertos. Foram feitos vários testes com diferentes tamanhos de janela. A Tabela 10 apresenta todos os resultados dos diferentes testes feitos nos algoritmos LRU e LRU com janela.

Foram testados diferentes tamanhos de janela de 1 bloco (6.3 MB) a 5000 blocos (3.07GB) para ver se ocorreu uma modificação nos resultados, mas foi encontrado uma variação nas taxas de acertos para janela abaixo de 30 blocos (18.9 MB). Há um limite para tamanho da janela, que seria o tamanho do filme, pois a janela avalia somente os clientes dentro do mesmo filme e uma janela extremamente grande seria prejudicada nas partes finais do filme, já que o número de blocos finais do filme poderia ser menor que o tamanho da janela selecionada.

Os resultados das simulações encontradas na Tabela 10 demostram que LRU Janela teve uma taxa de acerto inferior ao LRU normal. Entretanto, taxa de acertos tem uma melhora de acordo com crescimento da janela e depois o valor se estabiliza, no entanto, esses valores ainda ficam abaixo dos valor do LRU normal. Caso se inclua o valores de desvio padrão da Tabela 7, mesmo assim os valores da taxa de acerto do LRU Janela ficam abaixo do LRU normal.

Após uma reflexão sobre os resultados produzidos pelo LRU com janela, concluímos que usar esse algoritmo para *proxy* não é benéfico, pois ele da prioridade para blocos recém acessados que nem sempre serão reusados. Também temos de levar em conta que a janela limita o valor máximo da pontuação do bloco, tem uma ocorrência grande de números repetidos e sem outro critério de desempate a não ser a ordem de chegada, causando essa pontuação inferior ao LRU normal. Na próxima seção, teremos diferentes simulações com diferentes cenários para ajudar a comprovar o desempenho inferior do LRU com janela.

Tabela 10 – Média da taxa de acertos LRU versus LRU com janela

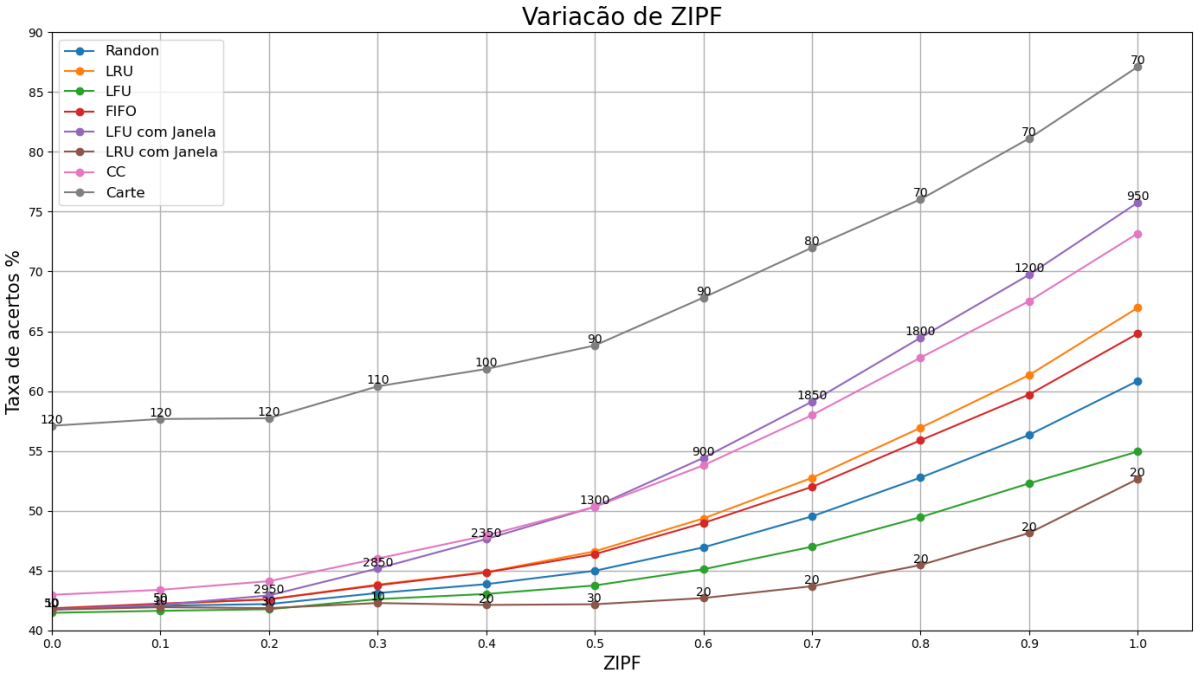
Janela	*LRU com Janela*	*LRU*	*Resultado*
1	42,3110%	44,085%	-1,774%
10	42,2951%	44,085%	-1,790%
20	42,2838%	44,085%	-1,801%
27	42,3218%	44,085%	-1,763%
28	42,3198%	44,085%	-1,765%
29	42,3162%	44,085%	-1,769%
30	42,3162%	44,085%	-1,769%
40	42,3162%	44,085%	-1,769%
50	42,3162%	44,085%	-1,769%
100	42,3162%	44,085%	-1,769%
150	42,3162%	44,085%	-1,769%
200	42,3162%	44,085%	-1,769%
250	42,3162%	44,085%	-1,769%
500	42,3162%	44,085%	-1,769%
1500	42,3162%	44,085%	-1,769%
3000	42,3162%	44,085%	-1,769%
5000	42,3162%	44,085%	-1,769%

Fonte: Autoria própria

5.3 Resultados de LFU versus LFU com janela

Os resultados produzidos pelo LFU com janela se mostraram bem mais promissores em alguns cenários. Foram feitas uma bateria de testes com variação de alguns dados da Tabela 8 e Tabela 9 que serão demonstrados abaixo:

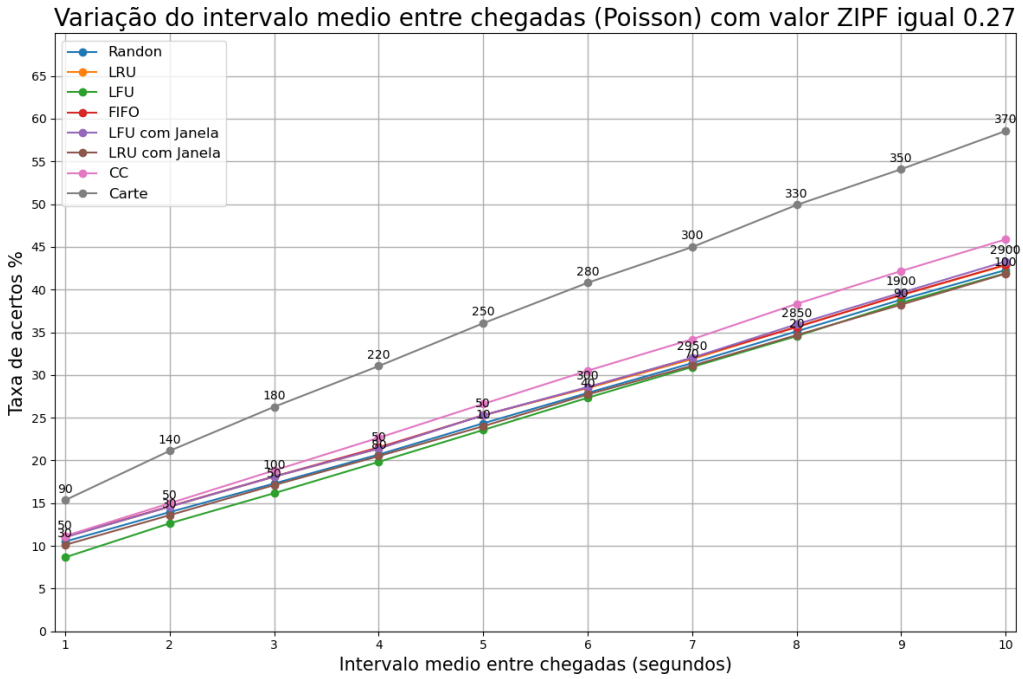
Figura 21 – Taxa de acertos com a variação de Zipf diversos algoritmos



Fonte: Autoria própria

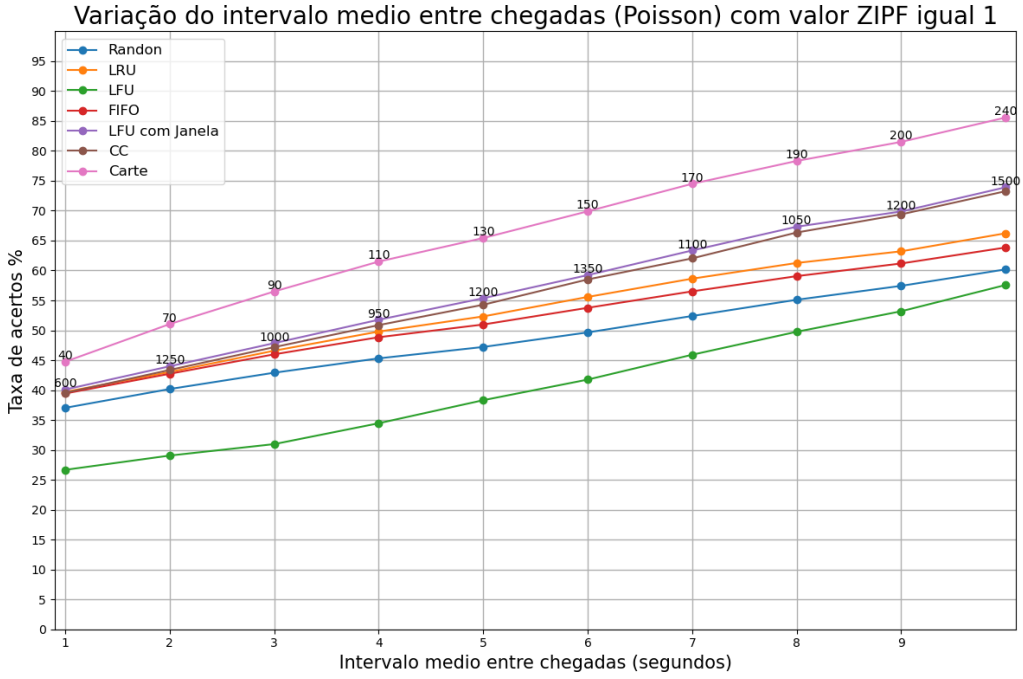
Os dados informados na Figura 21 mostram uma taxa de acerto mais alta à medida que o valor de Zipf aumenta, mostrando que LFU com janela tem um melhor funcionamento com valores altos de Zipf. O algoritmo de substituição LFU com janela tem uma grande melhoria em relação a outros algoritmos, como CC, LRU, FIFO, Randon e LFU. Cabe salientar que os valores são uma média da taxa de acerto para diversas simulações sobre o mesmo cenário, mas com base nos valores de desvio padrão e variação da tabela 7, estima-se que o ganho produzido pelo LFU com janela com Zipf seja na faixa de 20% em comparação com o LFU normal. Nos gráficos estão presentes o valor da janela que trouxe uma maior taxa de acertos dentro da faixa de variação descrito na seção 5.1.

Figura 22 – Taxa de acertos com a variação do intervalo médio chegada (Poisson) com ZIPF 0.27 para diversos algoritmos de substituição



Fonte: Autoria própria

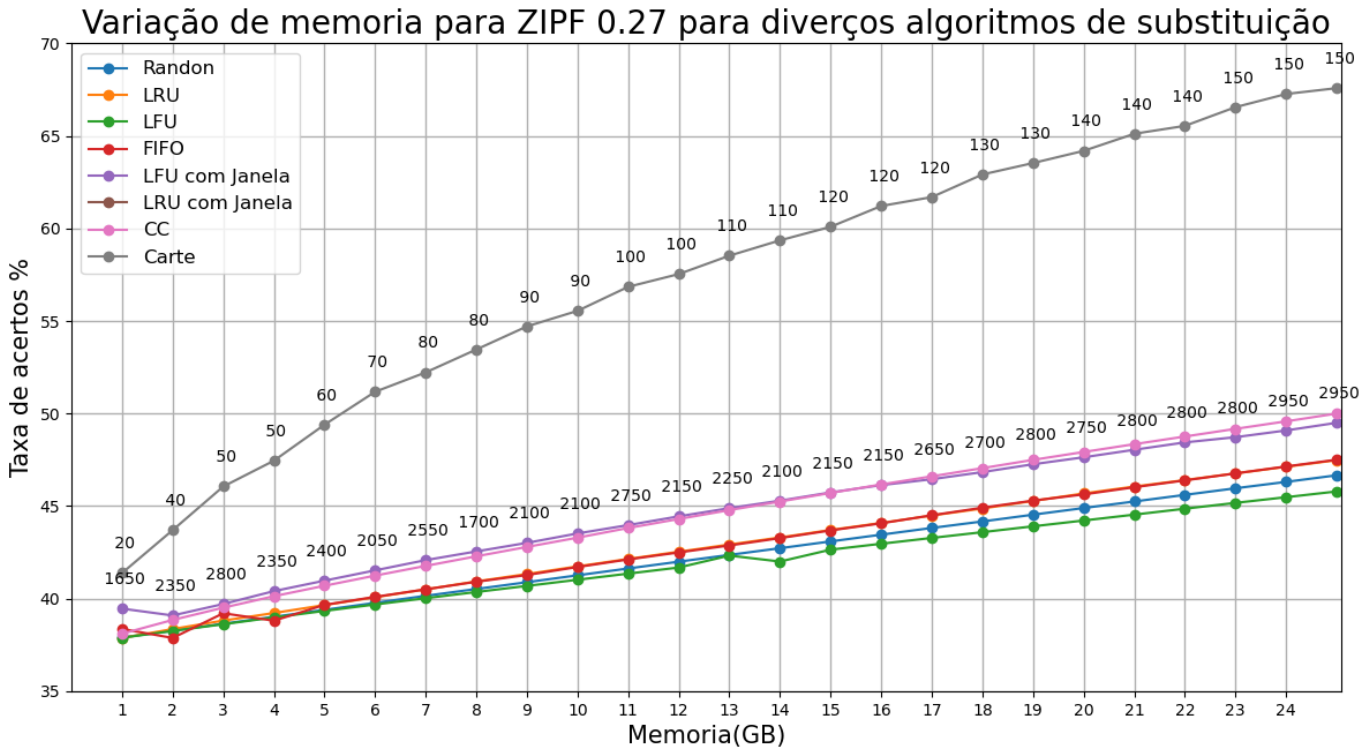
Figura 23 – Taxa de acertos com a variação do intervalo médio chegada (Poisson) com ZIPF 1 para diversos algoritmos de substituição



Fonte: Autoria própria

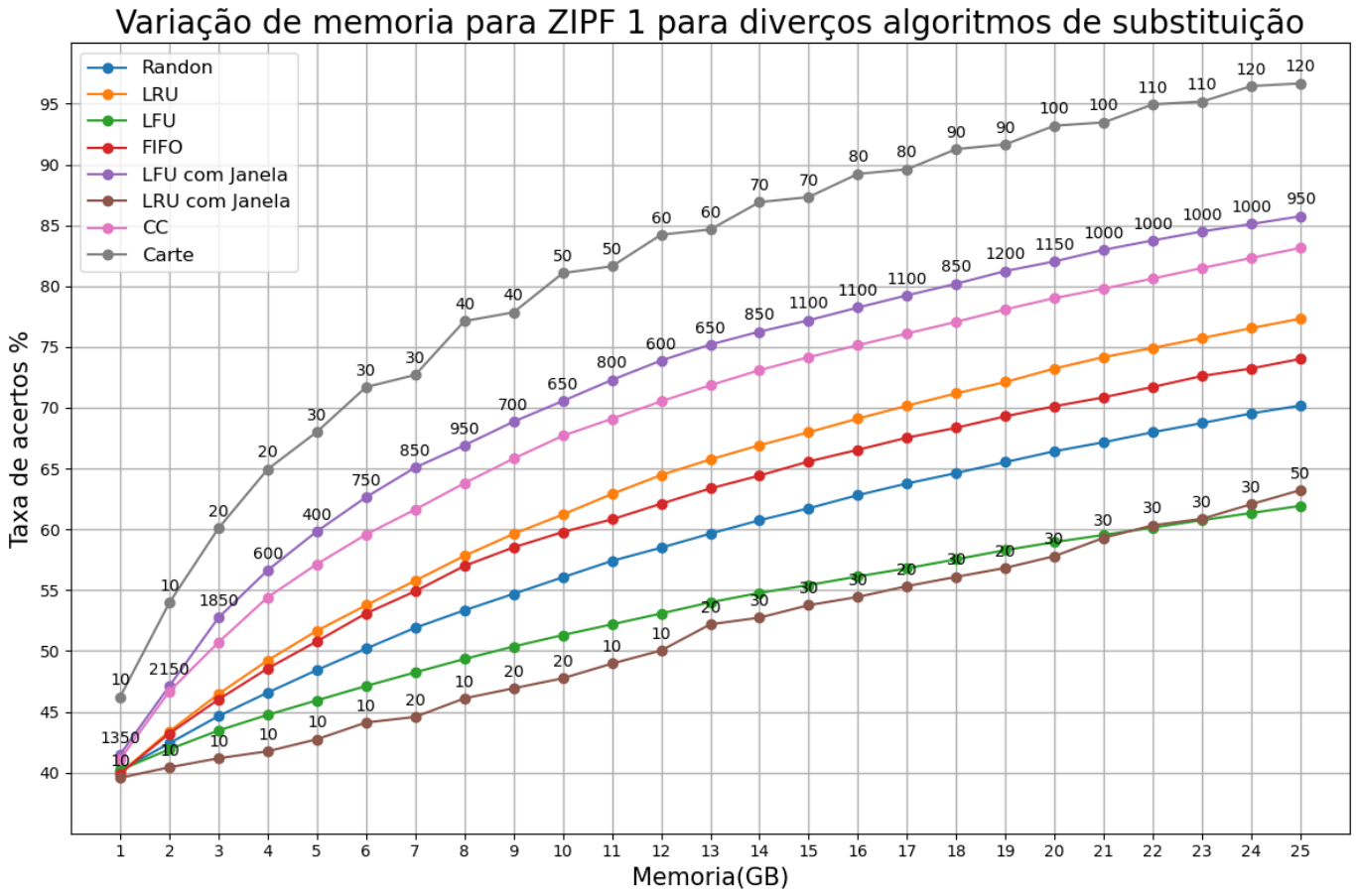
Com o resultado obtido através do experimento da Figura 21, partimos para os experimentos para variação do intervalo médio da chegada (Poisson) com cenários com Zipf alto e baixo que tem os resultados demonstrados na Figuras 22 e 23, lembrando que Zipf alto foi observado em sistemas de vídeo sob demanda real, conforme mencionado na seção 2.2. Ao analisar as Figuras 22 e 23, percebemos que variação do intervalo médio de chegada (Poisson) também influencia a taxa de acerto. No entanto, esse aumento na taxa de acertos influencia muito pouco no distanciamento LFU e LFU com Janela.

Figura 24 – Taxa de acertos com a variação de memória com ZIPF 0.27 para diversos algoritmos de substituição



Fonte: Autoria própria

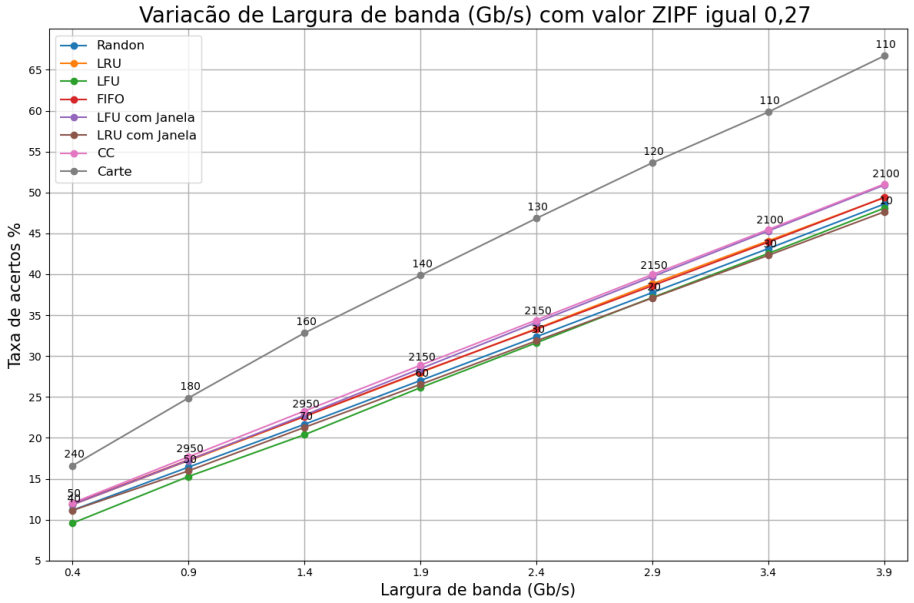
Figura 25 – Taxa de acertos com a variação de memória com ZIPF 1 para diversos algoritmos de substituição



Fonte: Autoria própria

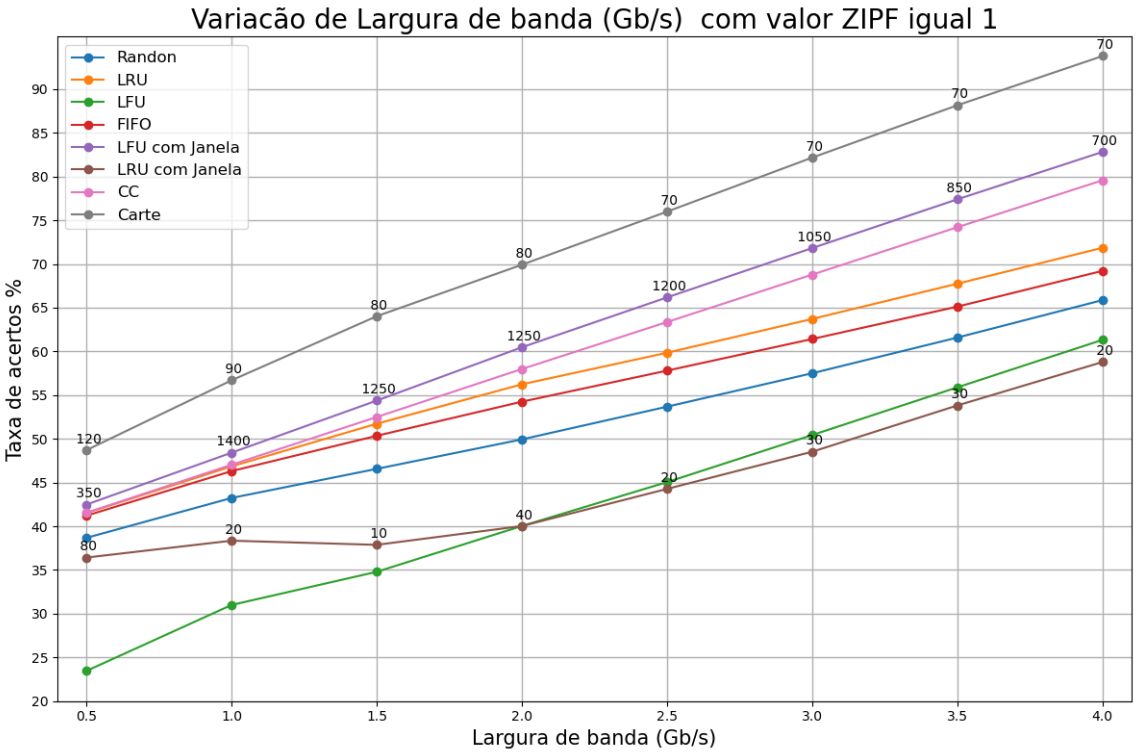
Partimos para o próximo experimentos de variação de memória com cenários de Zipf alto e baixo. Ao analisar as Figuras 24 e 25 é perceptível que a varição de tamanho da memória também influencia na taxa de acerto, uma vez que quanto maior é a memória, maior é a taxa de acerto. O valor máximo utilizado de 25 GB usado corresponde a menos de 8% do tamanho total do acervo utilizado, que pode ser calculado multiplicando 100 vídeos pelo tamanho padrão de cada vídeo que é 3.4GB. Caso o servidor *proxy* tivesse os 340GB de memória, teria uma taxa de acerto igual de 100%, que é inviável devido ao alto custo que esta configuração acarretaria, e também pode considerar um desperdício de recursos, pois não há necessidade que todos os arquivos estejam no mesmo momento no *proxy*, pois o comportamento dos usuários sobre os conteúdos são muito oscilantes.

Figura 26 – Taxa de acertos com a variação de largura de banda com ZIPF 0.27 para diversos algoritmos de substituição



Fonte: Autoria própria

Figura 27 – Taxa de acertos com a variação de largura de banda com ZIPF 1 para diversos algoritmos de substituição



Fonte: Autoria própria

Nas Figuras 26 e 27 mostram os resultados da taxa de acerto com a variação da largura de banda de rede entre o *proxy* e o servidor principal. A rede do *proxy* é importante, pois ela é responsável pelo intercâmbio entre o cliente e o servidor principal, permitindo uma renovação do conteúdo. Uma renovação frequente do conteúdo permite uma maior taxa de acertos, pois consegue disponibilizar os conteúdos mais atuais na cache que possivelmente serão mais acessados.

O aumento da memória e da largura de banda da rede implicam diretamente a taxa de acertos, pois são dois fatores que aumentam a eficiência do servidor *proxy* em termos de quantidade de clientes atendidos. Como a taxa de acertos é calculada através do total de pacotes entregue pelo servidor e isso inclui os pacotes que o servidor *proxy* teve que solicitar usando sua banda da rede.

Analisando os dados apresentados nas Figuras 21, 22, 23, 24, 25, 26 e 27, o LFU com janela sobressai o LFU nesses diferentes cenários. Apesar disso, a aplicação da janela no LFU não permitiu ainda ganhar do algoritmo CARTE, mas demonstrou que a aplicação da janela trouxe uma melhora significativa na taxa de acertos no algoritmo de substituição LFU. No entanto, ainda é necessário fazer várias baterias de teste para conseguimos ver o desempenho do LFU com janela em outras situações e com configurações diferentes.

Os resultados encontrados do algoritmo CARTE são resultados semelhantes encontrados no trabalho do Neves (2015). No entanto esse algoritmos CARTE é uma adaptação do CARTE original, mesmo assim trouce taxa de acerto parecidos e o tamanho de janela muito próximos. Esse valores das baterias testes CARTE e CC podem confirmar a validação do simulador, pois ao valores encontrados são parecidos ao trabalho do Neves (2015)

Além do resultado obtido, temos ainda muito a pesquisar, como a variação da janela automaticamente. Com isso, os algoritmos poderão ficar variando o tamanho da janela de acordo com o comportamento das requisições de vídeo, isso pode causar ganhos para Carte e para LFU com janela.

Nas Figuras 21, 22, 23, 24, 25, 26 e 27 temos LRU com Janela, mostrando um desempenho inferior com os outros algoritmos dos gráficos. Conseguimos comprovar através dessa bateria de testes, que esse algoritmo não trouxe resultado satisfatório para a aplicação do *proxy* nos cenários simulados. No entanto, o LFU com Janela demostrou um bom desempenho, mas precisamos de mais baterias de testes para podermos comprovar o desempenho desse algoritmo em diversos cenários.

Essas janelas usadas nos algoritmos LRU com Janela, LFU com Janela e CARTE,

podem ser aplicadas a outros algoritmos existentes que não foram abordados neste trabalho. Aplicação desse recurso a outros algoritmos podem produzir bons ou maus resultados, como ocorreu com LRU com Janela e LFU com Janela, entretanto, é um recurso promissor em termos de aplicabilidade em outros algoritmos.

6 CONSIDERAÇÕES FINAIS

Os servidores *proxies* VoD se tornam importante, pois mantém cópia do conteúdo acessado e consequentemente as requirições do cliente não necessitam trafegar até o servidor principal, reduzindo o tráfego da rede em alguns níveis. Um bom algoritmos de substituição para os servidores *proxies* VoD ajuda a melhorar o desempenho aumentando a taxa de acerto.

O algoritmo LRU com a Janela apresentou um desempenho pior ou igual ao LRU normal na função de algoritmos de substituição. Enquanto o algoritmo LFU com a Janela mostrou um promissor desempenho em comparação ao LFU normal, principalmente quando o número Zipf é alto, chegando a uma diferença de 20% entre os algoritmos. O LFU com a Janela e CC mostram desempenho semelhante, seu desempenho pode ser igualado quando levamos em conta os valores de desvio padrão de 1,6% e variação entre menor e maior de 5.30%.

Na bateria de testes realizada, o algoritmo CARTE mostrou um desempenho melhor comparado com os demais algoritmos analisados, no trabalho Neves (2015) também demostrou um desempenho semelhantes ao encontrado. Apesar de ser um algoritmo adaptado para funcionar no simulador, manteve com um desempenho superior a 5% em todos os testes. No entanto, deve ser realizado mais testes para verificar se o algoritmo LFU com a Janela não possa chegar a um despenho igual ou superior ao CARTE em alguns cenários com Zipf mais altos.

Conclui-se que os resultados encontrados demonstram que a aplicação de uma janela de tempo em conjunto com algoritmos de substituição pode ser promissora, contudo há necessidade de se realizar mais estudos com outros diferentes cenários de carga para observar mais amplamente os efeitos produzidos e a efetividade dos resultados. Além disso, considera-se a possibilidade de tornar o ajuste da janela dinâmica para conseguir uma melhor taxa de acerto frente a cenários de flutuação das condições de carga do sistema. Deve-se pensar analogamente também que, ao se modificar as configurações do *proxy*, isto pode igualmente proporcionar um impacto nos resultados.

REFERÊNCIAS

ALMEIDA, J. M. et al. Analysis of educational media server workloads. In: **Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video**. New York, NY, USA: Association for Computing Machinery, 2001. (NOSSDAV '01), p. 21–30. ISBN 1581133707. Disponível em: https://doi.org/10.1145/378344.378348.

BHAT, D. et al. Sabr: Network-assisted content distribution for qoe-driven abr video streaming. **ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)**, ACM New York, NY, USA, v. 14, n. 2s, p. 1–25, 2018.

BRIZOLA, J.; FANTIN, N. Revisão da literatura e revisão sistemática da literatura. **Revista de Educação do Vale do Arinos-RELVA**, v. 3, n. 2, 2016.

BUYYA, R.; PATHAN, M.; VAKALI, A. **Content delivery networks**. [S.l.]: Springer Science & Business Media, 2008. v. 9.

CHEN, X. et al. Hit ratio driven mobile edge caching scheme for video on demand services. In: IEEE. **2019 IEEE International Conference on Multimedia and Expo (ICME)**. [S.l.], 2019. p. 1702–1707.

CISCO. **Icons for PowerPoint de Network Topology Icons**. 2016. Disponível em: https://www.cisco.com/c/en/us/about/brand-center/network-topology-icons.html.

CISCO. **Cisco Annual Internet Report (2018–2023) White Paper**. San José, Califórnia, EUA, 2020.

CULLEN, C.; CANTOR, L. **The Global Internet Phenomena Report COVID-19 Spotlight**. Sandvine, EUA, 2020.

DAN, A.; SITARAM, D.; SHAHABUDDIN, P. Dynamic batching policies for an on-demand video server. **Multimedia systems**, Springer, v. 4, n. 3, p. 112–121, 1996.

Desmouceaux, Y. et al. A content-aware data-plane for efficient and scalable video delivery. In: **2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)**. [S.l.: s.n.], 2019. p. 10–18.

ELETRICISTAS, I. d. E. d. R. Instituto Americano de E. Ieee xplore. 1963.

FOECKLER VICTOR HENNING, J. R. P. **Mendeley**. 2008. Disponível em: www.mendeley.com.

GAGNE, A. S. . P. B. G. . G. (Ed.). **Fundamentos de sistemas operacionais**. Rio de Janeiro: Grupo GEN LTC, 2015.

GHOREISHI, S. E. **Bring the Content Closer to the End User: In-Network Adaptation and Caching of Mobile Video**. [S.l.], 2017. Disponível em: https://kclpure.kcl.ac.uk/portal/files/103728488/2017 *GhoreishiSeyedEhsan*1239776.

HAMMING, R. Acm. 1947.

HAO, H. et al. Knowledge-centric proactive edge caching over mobile content distribution network. In: IEEE. **IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)**. [S.l.], 2018. p. 450–455.

HEFEEDA, M.; SALEH, O. Traffic modeling and proportional partial caching for peer-to-peer systems. **IEEE/ACM Transactions on networking**, IEEE, v. 16, n. 6, p. 1447–1460, 2008.

HONG, D.; VLEESCHAUWER, D. D.; BACCELLI, F. A chunk-based caching algorithm for streaming video. In: . [S.l.: s.n.], 2010.

IKEDA, Y. et al. An enhancement of efficiency for robust incomplete factorization preconditioning based upon a-orthogonalization process. **Trans. of IPSJ**, v. 45, 2004.

JAYAREKHA, P.; NAIR, T. An adaptive dynamic replacement approach for a multicast based popularity aware prefix cache memory system. **arXiv preprint arXiv:1001.4135**, 2010.

KAREDLA, R.; LOVE, J. S.; WHERRY, B. G. Caching strategies to improve disk system performance. **Computer**, IEEE, v. 27, n. 3, p. 38–46, 1994.

KOCH, C. et al. Category-aware hierarchical caching for video-on-demand content on youtube. In: **Proceedings of the 9th ACM Multimedia Systems Conference**. New York, NY, USA: Association for Computing Machinery, 2018. (MMSys '18), p. 89–100. ISBN 9781450351928. Disponível em: https://doi.org/10.1145/3204949.3204963.

LEE, M.-C. et al. Throughput–outage analysis and evaluation of cache-aided d2d networks with measured popularity distributions. **IEEE Transactions on Wireless Communications**, IEEE, v. 18, n. 11, p. 5316–5332, 2019.

NEIVA, F. W. **Revisão Sistemática da Literatura em Ciência da Computação Um Guia Prático**. Tese (Doutorado) — Universidade Federal de Juiz de Fora, 2016.

NEVES, B. S. Proposta de algoritmo de cacheamento para proxies vod e sua avaliação usando um novo conjunto de métricas. 2015.

OSMAN, A. M.; OSMAN, N. I. A comparison of cache replacement algorithms for video services. **International Journal of Computer Science & Information Technology**, v. 10, n. 2, p. 95–111, 2018.

Pal, A.; Kant, K. Nacid: A neighborhood aware caching and interest dissemination in content centric networks. In: **2017 26th International Conference on Computer Communication and Networks (ICCCN)**. [S.l.: s.n.], 2017. p. 1–9.

PATTERSON, D. A.; HENNESSY, J. L.; ASHENDEN, P. J. **Computer Organization and Design: The Hardware**. [S.l.]: Elsevier Science Limited, 2007.

SOARES, M. V. B. et al. Pretext ii: descrição da reestruturação da ferramenta de pré-processamento de textos. São Carlos, SP, Brasil., 2008.

SPRINGER, J. Springer link. 1842.

TAHER, S. et al. A Review on Cache Replacement Strategies in Named Data Network . **researchgate.net**, 2018. ISSN 2289-8131. Disponível em: https://www.researchgate.net/publication/326331089.

VERSTAK, A. A. A. Google scholar. 2004.

WANG, K.; CHEN, F. Cascade mapping: Optimizing memory efficiency for flash-based key-value caching. In: **SoCC 2018 - Proceedings of the 2018 ACM Symposium on Cloud Computing**. New York, NY, USA: Association for Computing Machinery, Inc, 2018. p. 464–476. ISBN 9781450360111. Disponível em: https://doi.org/10.1145/3267809.3267847.

WHITE, C. M. et al. **Video on demand methods and systems**. [S.l.]: Google Patents, 2008. US Patent App. 12/101,793.

WILLIAM, S. **Arquitetura e Organização de Computadores**. [S.l.]: São Paulo, BR: Pearson Pratice Hall, 2010.

APÊNDICE A – DOCUMENTO ESPECIFICAÇÃO E DETALHAMENTO SIMULADOR

Documento Especificação e Detalhamento
Simulador

Matheus Koch

Dezembro 2020

Sumário

1 Introdução

1.1 Propósito de um simulador de Proxy VoD

Gerar relatórios do desempenho do uso de diferentes algoritmos de cacheamento proxy VoD e permitir adicionar diferentes algoritmos de maneira fácil para fazer as simulações. Público alvo: usuários que necessitem de ambiente de testes desempenhando algoritmos de cacheamento proxy VoD.

1.2 Escopo do produto

Terá que gerar resultados do proxy Vod. O ambiente proxy VoD terá as seguintes características:

- Não terá bloco de vídeo reais.
- O tempo dentro do simulador passará diferente do tempo real.
- Instante de entrada dos clientes sera dado na leitura de arquivo de texto.
- Os parâmetros do Proxy Vod serão definidos como memória e limite de banda.
- Não terá que fazer conexões externas ao ambiente de rede.
- Segundo do simulador passará quando ele terminar de executar todas as tarefas propostas dentro do segundo.
- Não terá limite de entrada limites.

1.3 Convenções, termos e abreviações

A correta interpretação deste documento exige o conhecimento de algumas convenções e termos específicos, os quais que são descritos a seguir: VoD - Video on Demand proxy- servidor proxy VoD

1.4 Propriedades dos requisitos

Para estabelecer a prioridade dos requisitos, nas seções 4 e 5, foram adotadas as denominações essencial, importante e desejável.

Essencial é o requisito sem o qual o sistema não entra em funcionamento. Requisitos essenciais são requisitos imprescindíveis, que têm que ser implementados impreterivelmente.

Importante é o requisito sem o qual o sistema entra em funcionamento, mas de forma não satisfatória. Requisitos importantes devem ser implementados, mas, se não forem, o sistema poderá ser implantado e usado mesmo assim.

Desejável é o requisito que não compromete as funcionalidades básicas do sistema, isto é, o sistema pode funcionar de forma satisfatória sem ele. Requisitos desejáveis podem ser deixados para versões posteriores do sistema, caso não haja tempo hábil para implementá-los na versão que está sendo especificada.

2 Descrição geral do sistema

Será um software de simulador de rede proxy VoD, tendo como objetivo principal retornar os resultados dos empregos de algoritmos de cacheamento. Ele receberá um arquivo, no qual será lido para gerar clientes, sendo que no arquivo terá o tempo no qual o cliente entrar. Como se trata de uma simulação, a contagem do tempo pode ser feita de formas diferentes, pois um segundo do simulador pode ser menor que um segundo real. Isso é bom de se fazer, a fim de conseguir resultados mais rápidos que na execução de um servidor proxy VoD real. O sistema terá o bloco bruto do vídeo e não será limitado pela memória que tiver o computador que está o executando.

3 Requisito Funcionais

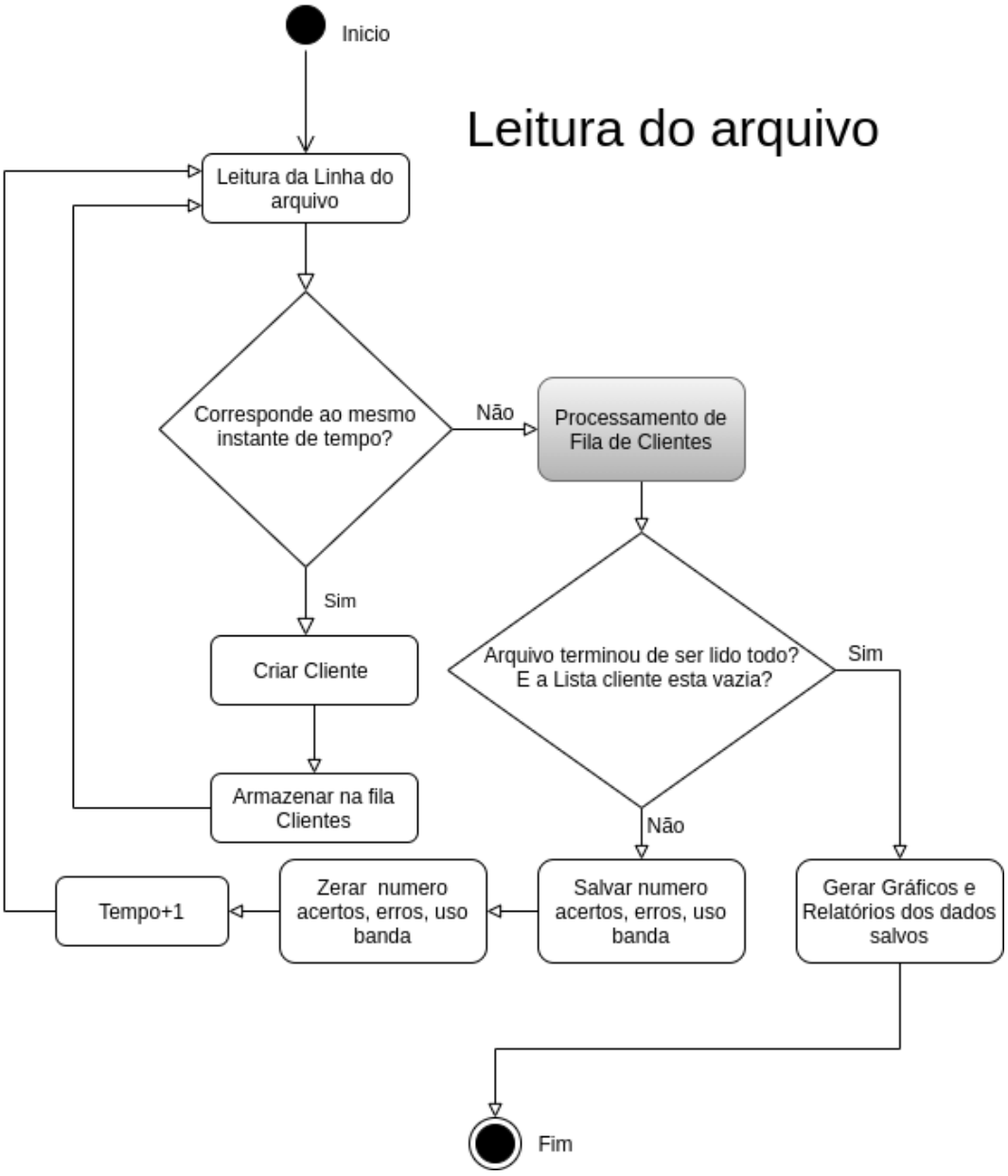
N°	Prioridade	Descrição
1	Essencial	Deve contabilizar o tempo dentro da simulação e cada segundo corresponde à entrega do bloco de vídeo de todos os clientes ativos, dentro do limite de banda e memória.
2	Essencial	Ler um arquivo com as entradas de clientes. O arquivo contém 4 colunas: id cliente, Filme, tempo e início do vídeo usando Zipf.
3	Essencial	Linha com informações de clientes lida no tempo certo. Deve gerar clientes.
4	Essencial	Quando o cliente terminar de receber todos os blocos do vídeo, o cliente é desconectado.
5	Essencial	A memória tem que receber o bloco do servidor principal.
6	Essencial	A cada segundo ele tem que ler o cliente e enviar o bloco de vídeo que ele precisa.
7	Essencial	Tem que armazenar e agrupar os pedidos de vídeo que seriam feitos para o servidor principal.
8	Essencial	Terá um limite de pedidos e requisições a serem atendidos.
9	Essencial	Distribuição dos blocos de vídeos requisitados atendidos.
10	Essencial	Os blocos substituídos da memória, ordenamos em uma fila por alguns algoritmos com suas lógica.

4 Requisitos Não-Funcionais

N°	Prioridade	Descrição
1	Essencial	Tempo da simulação executado seja menor que o tempo da simulação em tempo real.
2	Essencial	Facilidade em carregar algoritmos de substituição e configurar o ambiente de teste.

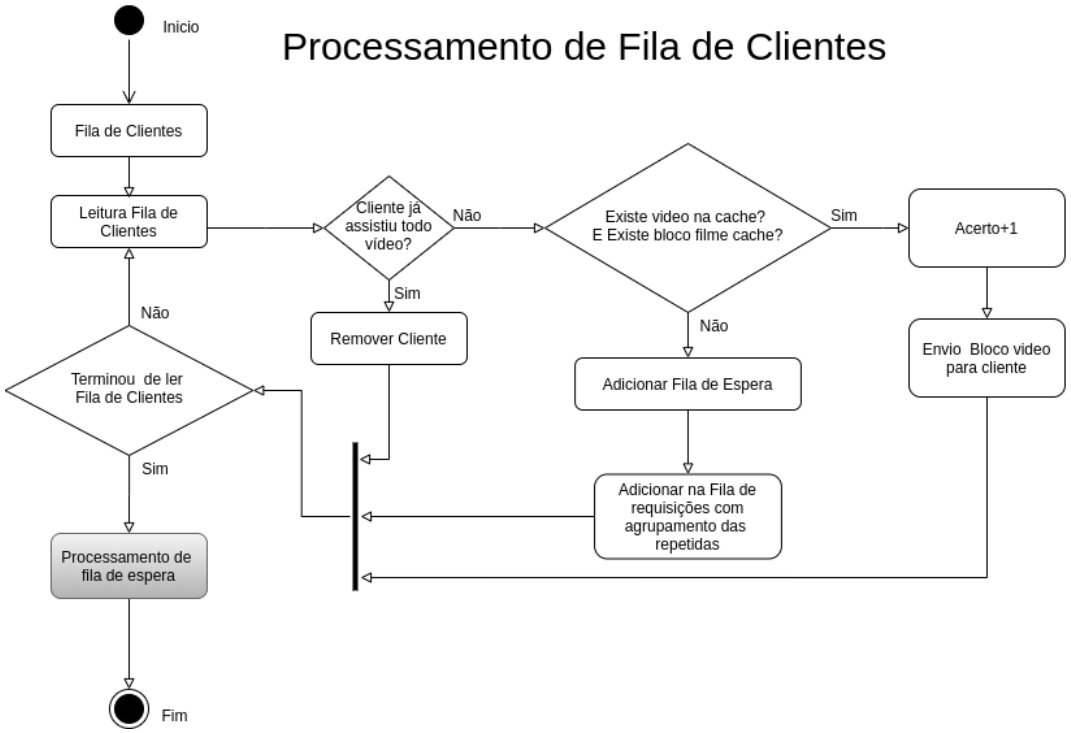
5 Diagramas Funcionamento

Figura 1: UML funcionamento da leitura do arquivo no simulador.



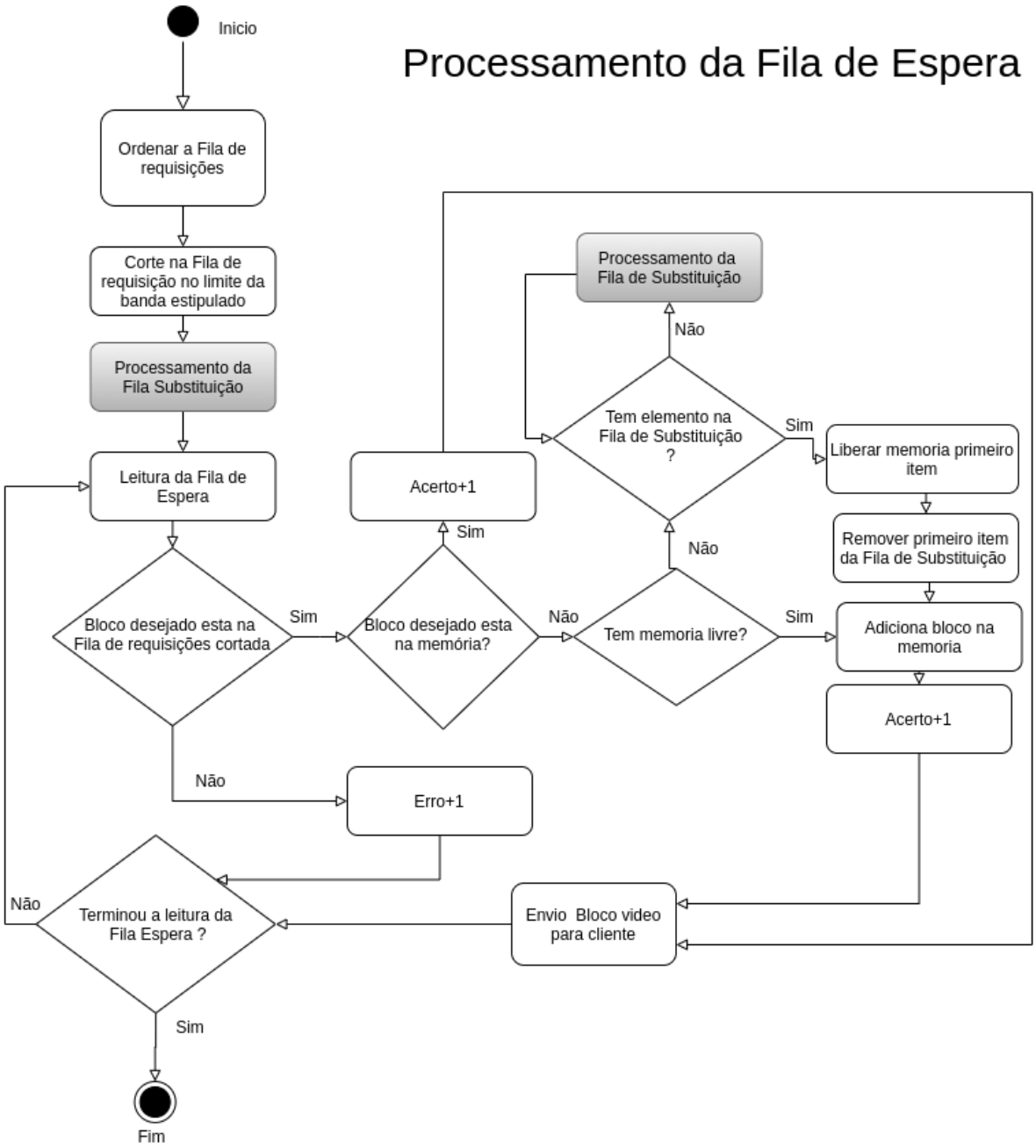
Autoria própria

Figura 2: UML funcionamento da processamento da fila de Clientes no simulador.



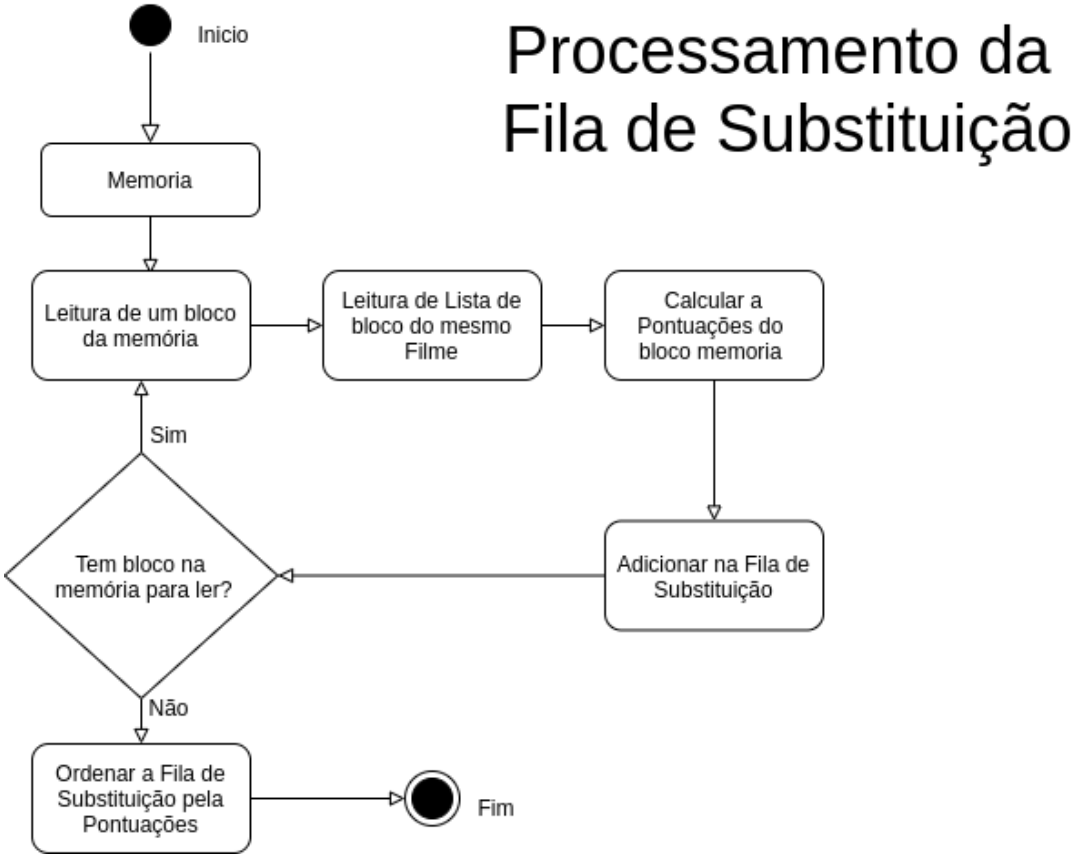
Autoria própria

Figura 3: UML funcionamento da processamento da fila de espera no simulador.



Autoria própria

Figura 4: UML funcionamento do processamento da fila substituição no simulador.



Autoria própria

6 Descrição das Funcionalidade

6.1 Contador Tempo

O tempo terá contador e acrescentará +1 a cada término de atendimento de todos os clientes, em que ele vai atender todos os clientes dentro dos recursos (memória e largura de banda) disponíveis naquele momento. Ao fim do segundo ele terá que salvar o contador de acertos, o contador de erros e o contador do uso de banda. Também terá que zerar as filas de espera, as filas de requisições, o contador de acertos, o contador erros e o contador do uso de banda.

6.2 Leitura de arquivo

Permite ler um arquivo e pode ter diferentes números de linhas. Cada linha vai ter 4 colunas com valores de inteiro separados por um espaço (”0 0 0 0”):

- Primeiro coluna, id cliente, id é único
- Segunda coluna, id filme, pode ser repetido
- Três coluna, tempo entrada,
- Quadro coluna, tempo de ficar acessado

Na terceira coluna o arquivo de leitura terá ordem de tempo correta com o tempo crescente, sem a necessidade de organizar todo o arquivo em ordem de tempo.

6.3 Adicionar cliente na Fila Clientes

Os clientes virão da leitura de um arquivo explicado no requisito RF001. A leitura será linha por linha e cada leitura será criado objeto cliente, com dados vindos da linha de leitura e depois armazenados na Fila de Clientes. Deve ler o arquivo até o instante que o cliente for um instante maior que o instante em que o Simulador está. Assim pode haver uma ou mais leituras, criações e armazenamentos de clientes no mesmo instante. O arquivo de leitura é gerado usando Zipf- intervalo médio da chegada de cliente.

6.4 Remover cliente na Fila Clientes

Após clientes após o bloco atualizar e for igual ao bloco final do vídeo, o cliente deve ser removido da Fila de Clientes. Isso deve ocorrer no início da leitura da Fila de clientes.

6.5 Adicionar bloco na Memoria

A Memória será representada por um conjunto de blocos de vídeos de 1 segundo, na qual os algoritmos de substituição terão que trabalhar sempre com esses blocos. A Memória será uma lista com blocos que terão informações sobre o identificador filme, identificador bloco, parâmetros é número de clientes. O argumento genérico receberá um valor dos algoritmos de substituição. Ao tentar adicionar Memória, tem que verificar se tem espaço, e se caso colocar, devem ser obedecidos os critérios anteriores de constituição da Memória. Caso não haja espaço, devem ser acionados os algoritmos de substituição, para receber a posição da Memória que será sobrescrita.

6.6 Leitura da Fila de Cliente

Os clientes devem verificar se existem blocos de vídeo que necessitam e, se encontrar, adicionar mais um no bloco de vídeo que o cliente está, ademais de contabilizar mais um contador de acertos. Caso não encontre o bloco deve entrar em Adicionar Fila de Requisições e Fila de Espera.

6.7 Adicionar Fila de Requisições

Uma fila de espera contará com clientes que não encontraram conteúdo no proxy ao mesmo tempo, para que ele seja solicitado no mesmo instante e entregue dentro dos limites da banda. Na criação dessa fila, também terá que adicionar o cliente da requisição na Fila de Espera.

6.8 Solicitar as requisições da Fila de Requisições

A ordem da Fila de Requisições obedecerá três critérios:

- Número de clientes que querem a mesma requisição;
- Número de clientes que querem a parte do mesmo vídeo;
- Ordem adicionada na Fila de Espera.

Depois de organizada, devem ser solicitadas as requisições que estiverem dentro da quantidade que limita a banda. As demais devem ser descartadas.

6.9 Leitura da Fila de Espera

Depois da solicitação de requisições, os clientes serão contemplados, assim deve-se adicionar mais um no bloco de vídeo que o cliente está, contabilizar mais um contador de acertos e também terá que adicionar ou substituir o bloco de vídeo na memória. Depois dos clientes terá que adicionar mais um no bloco de vídeo que o cliente está e contabilizar mais um contador de erros. Depois disso pode prosseguir para [RF004] Contador Tempo.

6.10 Algorismo de Substituição

Antes da leitura da Fila de Espera, sempre é sera gerado um File de Substituição, nessa fila tera informações da posição da memoria e sua pontuação, essa pontuação dever ser calculado pelo Algorismo de Substituição. Durante leitura da Fila de Espera ele fila sera consumida para colocar novos bloco na memoria. Caso acabe essa fila de Substituição, ela recalcula novamente a pontuação dos blocos da memoria.